NETWORK CODING ENABLED NAMED DATA NETWORKING ARCHITECTURES

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Jonnahtan Eduardo Saltarin De Arco

von Venezuela

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

NETWORK CODING ENABLED NAMED DATA NETWORKING ARCHITECTURES

Inauguraldissertation
der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Jonnahtan Eduardo Saltarin De Arco

von Venezuela

Leiter der Arbeit:
Professor Dr. Torsten Braun
Institut für Informatik

Von der Philosophisch-naturwissenschaftlichen Fakultät angenommen.

Bern, November, 2017

Der Dekan:
Prof. Dr. Gilberto Colangelo

Dedicated to my mother…

Dedicada a mi madre…

Acknowledgements

The work here presented was carried out as part of my doctoral studies at the University of Bern, and it would not have been possible without the guidance and support of many people to whom I would like to express my gratitude.

Foremost, I would like to thank my thesis supervisor, *Prof. Dr. Torsten Braun*, for giving me the opportunity to join the Communications and Distributed Systems lab (CDS) of the University of Bern, and for guiding me towards the achievement of the doctoral degree. I am grateful for all the advice, ideas and corrections you have given me, and which helped me publish my work and complete this thesis. I would also like to thank you for allowing me the freedom to pursue my own ideas during my doctoral studies, in the frame of the Swiss National Science Foundation project "Network Coding Based Multimedia Streaming in Content Centric Networks".

I would also like to express my deepest gratitude to *Prof. Dr. Nikolaos Thomos (Nikos)*, for all the patience, support, ideas and motivation he has given me. Thank you for all the detailed comments and corrections of the papers and this thesis. I have learned a lot of my writing skills from you. Moreover, without your encouragement during this four years, the way towards the Ph.D. would have been much more difficult. Many thanks also for participating in the preparation of the Swiss National Science Foundation project "Network Coding Based Multimedia Streaming in Content Centric Networks", in which I worked during my doctoral studies.

Many thanks also to *Dr. Eirini Bourtsoulatze*, who was always there to discuss interesting ideas and propose interesting points of view. Your attention to detail saved this thesis and our previous publications from many mistakes. Thanks!

I would like to extend my gratitude to the Swiss National Science Foundation, which funded my doctoral studies under grant number 149225.

A big thanks goes to all my CDS colleagues, who have contributed to my good time at the University of Bern. *Ali*, thanks for being a nice officemate, always ready to help. *José*, thanks for your camaraderie, the nice talks in Spanish, and for being my

gym companion. *Dima*, *Haider*, *Islam*, *Eirini B.*, *Imad*, *José*, thanks for bringing food from home and making the lunches a nice time. Many thanks to *Eryk* and *André* for having our computer infrastructure in good shape. *João*, *Carlos*, *Zhongliang*, *Zan*, *Eirini K.*, *Mikael*, *Mustafa*, *Matthias*, *Florian*, *Almerima*, *Desislava*, thanks for the good conversations, the advices and for making my time at CDS a wonderful time, even in the most difficult times. Thank you all!

Special thanks to *Daniela Schroth*. Thanks for your disposition to help me with all the administrative procedures from the university and beyond. I really have to acknowledge your patience in times when I went to your office full of questions and requests for help. Thanks also for being my voice and writer at times when I needed to communicate in German. Merci vilmal!

To my friend, *Francisco*, thanks for encouraging me during the most difficult times as a doctoral students, and for the good friendship you have always shared with me.

I would like to thank my family for all their support. This has to be in Spanish. Gracias mamá, por tu amor incondicional hacia mí, y por darme las bases que me han permitido obtener este logro. Gracias por los incontables sacrificios que hiciste para ayudarme a llegar a donde estoy. Gracias tambien a mis tías y tíos, siempre dispuestos a ayudarme cariñosamente. No podría estar aquí si no fuese por ustedes.

Last but not least, my best friend and soul-mate, *Ana*. You are my most important source of encouragement, knowledge, and love. Thanks for always sticking by my side, even in the most complicated times, academically and personally. Thanks also for the technical discussions we had and the writing advice that helped me a lot during this doctoral journey.

Bern, 15 September 2017 J. Saltarin

Abstract

The volume of data traffic in the Internet has increased drastically in the last years, mostly due to data intensive applications like video streaming, file sharing, *etc.*. This motivates the development of new communication methods that can deal with the growing volume of data traffic. To this aim, Named Data Networking (NDN) has been proposed as a future Internet architecture that changes how the Internet works, from the exchange of content between particular *nodes* of the network, to retrieval of particular *content* in the network. The NDN architecture enables ubiquitous in-network caching and naturally supports dynamic selection of content sources, characteristics that fit well with the communication needs of data intensive applications. However, the performance of data intensive applications is degraded by the limited throughput seen by applications, which can be caused by *(i)* limited bandwidth, *(ii)* network bottlenecks and *(iii)* packet losses. In this thesis, we argue that introducing *network coding* into the NDN architecture improves the performance of NDN-based data intensive applications by alleviating the three issues presented above. In particular, network coding *(i)* enables efficient multipath data retrieval in NDN, which allows nodes to aggregate all the bandwidth available through their multiple interfaces; *(ii)* allows information from multiple sources to be combined at the intermediate routers, which alleviates the impact of network bottlenecks; and *(iii)* enables clients to efficiently handle packet losses. This thesis first provides an architecture that enables network coding in NDN for data intensive applications. Then, a study demonstrates and quantifies the benefits that network coding brings to video streaming over NDN, a particular data intensive application. To study the benefits that network coding brings in a more realistic NDN scenario, this thesis finally provides a caching strategy that is used when the in-network caches have limited capacity. Overall, the evaluation results show that the use of network coding permits to exploit more efficiently available network resources, which leads to reduced data traffic load on the sources, increased cache-hit rate at the in-network caches and faster content retrieval at the clients. In

particular, for video streaming applications, network coding enables clients to watch higher quality videos compared to using traditional NDN, while it also reduces the video servers' load. Moreover, the proposed caching strategy for network coding enabled NDN maintains the benefits that network coding brings to NDN even when the caches have limited storage space.

Keywords: Named Data Networking (NDN), Network Coding, Video Streaming, Caching.

Contents

List of Figures

1

Introduction

1.1 Background

Nowadays, Internet users care more about the attributes of the content that they want to obtain (*e.g.*, name of the content, quality of the content, *etc.*) rather than where this content is located in the network. However, in the current Internet architecture, each packet is routed based on the network location of the destination host, without considering the attributes of the content that it carries. To address the mismatch between the users' behavior and the Internet architecture, Named Data Networking [38, 104] has been proposed. NDN is a new communication paradigm in which the importance is shifted from *where* the content is located, to *what* the content is. In the NDN architecture, the content is described by its *name*, which includes the content attributes (*e.g.*, for video content, it could include the video filename, quality, segment id, *etc.*). The users demand content with the help of *Interest* messages that contain the name of the requested content. The Interests are transmitted over the network until they reach a node holding a copy of the content object whose name matches that of the Interest. This node creates a *Data packet* that contains a copy of the requested

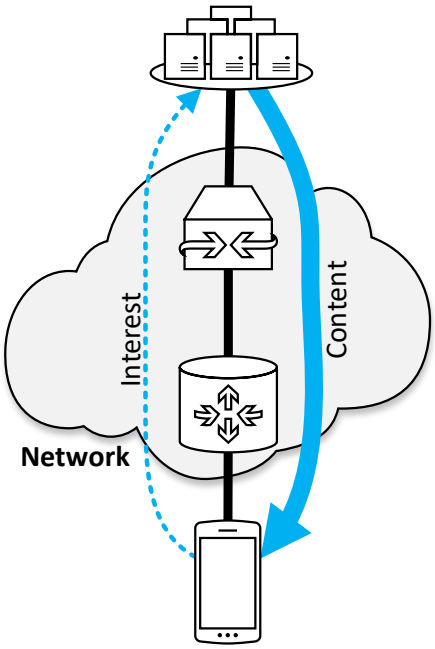

Figure 1.1: NDN content retrieval process.

content object and sends it back to the client. The Data packet follows the reverse path of that followed by the Interest. As the Data packet is transmitted backwards to the client, intermediate routers can cache copies of it, so they can reply to future Interests for the same content. An example of content retrieval in NDN is presented in Fig. 1.1

One of the main issues that future Internet architectures like NDN will need to face is the high amount of data traffic, mostly due to the high number of devices connected to the Internet and to the popularization of data intensive applications (*e.g.*, video streaming). Nowadays, data intensive applications are the major sources of content traffic on the Internet, and their predominance is expected to further increase in the near future [21]. To satisfy the expectations of the end-users (*e.g.*, receiving high quality video), these applications require a high and stable amount of throughput between the sources and the clients. However, the following issues may potentially degrade the throughput seen by the applications.

- *Limited bandwidth* on links connecting the network nodes may reduce the throughput seen by the clients to the point where it does not meet the requirements of data intensive applications. The use of multiple interfaces to retrieve content permits the network nodes to aggregate the bandwidth of their multiple

interfaces, increasing the throughput seen by the applications compared to a using a single interface [81]. In particular, clients with limited access bandwidth could use their multiple interfaces (*e.g.*, LTE, Wi-Fi, Bluetooth, *etc.*) to distribute the Interests needed to retrieve the desired content. In scenarios in which multiple clients in the network are interested in the same content objects (*e.g.*, a popular movie), and/or when the Data packets that compose a content object are distributed across multiple sources for load sharing (*e.g.*, in a multi-cloud storage system [84]), optimal multipath content retrieval is only attained if the Data packets are delivered over the optimal set of multicast trees [101]. The optimal set of multicast trees can only be computed by a central entity that is aware of the network topology and the content demands. However, this requires a high number of signaling messages to inform the central entity about topology and content demand changes in a timely manner. Hence, computing the optimal set of multicast trees is both computationally hard and difficult to implement in large dynamic networks.

- *Bottlenecks in the network* limit the flow of Data packets that is transmitted from the sources to the clients. In-network caching [24] and Interest aggregation [23] are two NDN characteristics that help to alleviate this issue, by reducing the number of Data packets transmissions in the network. However, in NDN, both in-network caching and Interest aggregation only work for exact name matching, which limits the scope of this solution.

- *Packet losses* in the network delay the content retrieval process since they require that clients detect which Data packet has not arrived in order to retransmit the corresponding Interest. In NDN, Interests have an expiration time, after which the client considers that the Interest will not be satisfied and can retransmit it. However, adapting the expiration time of each Interest to the current network conditions is a complicated task, since it should be short enough to detect packet losses in a timely manner, and long enough to allow a Data packet facing acceptable congestion to arrive without triggering a retransmission. Moreover, the expiration time of the Interests also depends on the requirements of the application that is requesting the content object. For example, a video streaming application requires low latency and, thus, it sets short Interest expire times, while a file download application for software updates may not need low latency, but prefers to avoid network congestion, thus, it sets long Interest expiration times.

1.2 Research Proposal

It is clear from Section 1.1 that content retrieval in NDN is problematic for data intensive applications, especially when multiple clients and/or multiple sources exist in the network. In this thesis we argue that introducing network coding [7] into the NDN architecture improves the performance of NDN-based data intensive applications, by alleviating the throughput loss for the applications due to limited bandwidth and inefficient content retrieval over multiple interfaces, network bottlenecks and packet losses.

Network coding has been proposed in previous works as a possible solution to the problems presented in Section 1.1, for host-centric Internet architectures [34, 101] and for content-centric Internet architectures [59, 100]. The main idea of network coding is to allow intermediate routers not only to forward Data packets, but also combine them before forwarding. This can potentially increase the throughput seen by the applications by allowing different Data packets of the same content object to use the full capacity of shared links [7] and facilitating multi-client and multi-source content retrieval via random network coding [34].

Unlike the original NDN architecture, in which clients sequentially request the Data packets that compose a content object, in a network coding enabled NDN architecture, the clients request a set of network coded Data packets that are generated by combining the Data packets that compose the requested content object. When network coding is enabled in NDN, the network coded Data packets contain information from all the Data packets that have been combined to generate them. Thus, all network coded Data packets with a specific name prefix are "equivalent" in terms of contained information. This reduces the granularity of the information and subsequently of the content requests, *i.e.*, in a network coding enabled NDN the Interests do not request specific Data packets but rather any network coded Data packet that belongs to the requested content object. Moreover, in a network coding enabled NDN architecture, the intermediate routers can apply network coding operations to the received Data packets before forwarding them, increasing the Data packet diversity in the network and allowing information belonging to the same content object that is stored in different sources to be combined into a single Data packet in the network. The clients can retrieve the original content object as soon as they receive a decodable set of network coded Data packets, *i.e.*, a set of Data packets that has as many linearly independent network coded Data packets as the number of Data packets composing the content

object.

As a result of enabling network coding in NDN, the nodes do not need to forward each Interest over the particular face that makes it travel over the optimal set of multicast trees, eliminating the need for coordination among clients. Instead, since the granularity of Data packets is reduced, the node could forward Interests over any face that is configured in to forward Interests with the requested name prefix, and still the data delivery will be optimal. This enables efficient multipath content retrieval, which enhances the bandwidth utilization. Moreover, since information from multiple sources can be combined by applying network coding at the intermediate nodes, the degradation of the throughput seen by the applications caused by network bottlenecks is alleviated. Additionally, the use of network coding simplifies the design of the clients, since the client does not need to keep track of the reception of each particular Data packet, but only the number of received innovative network coded Data packets. This means that clients can send an adequate number of Interests, taking into account the measured packet loss rate of the network, to receive the number of network coded Data packets they need to decode the requested content object, without any delay in case of packet losses.

1.3 Motivation

In the following subsections, we provide more details of the benefits that network coding brings to NDN, and illustrate them through motivating examples.

1.3.1 Efficient Multipath Content Retrieval

Nowadays, most client devices, *e.g.*, mobile phones, laptops, *etc.*, have two or more network interfaces over which they can receive the demanded content (*e.g.*, LTE, Wi-Fi, Bluetooth, *etc.*). However, in traditional host-centric networking, support for multipath is not extended. Recent efforts to support multipath communications on TCP (MP-TCP) [28] have been developed by the IETF. The drawback of these proposals is that they require end-to-end connections to be established for each host, which complicates load-balancing in the network and the dynamic selection of the sources. In comparison, NDN provides natural support for multipath content retrieval, without requiring end-to-end connections. This is achieved by allowing clients to distribute all the Interests needed to retrieve a video segment over all their available interfaces,

without knowing a priori which source or in-network cache can provide the particular Data packet that each Interest requests. However, despite having the necessary components for enabling multipath content retrieval, the original NDN architecture still lacks appropriate mechanisms for the optimal use of the multiple paths available for content retrieval in scenarios where multiple sources store different parts of the content (*e.g.*, in-network caches, content distributed in multiple clouds [84], *etc.*) and/or multiple clients are interested in the same content (*e.g.*, streaming popular videos, updating popular software, *etc.*).

Optimizing multipath content retrieval in multi-source and multi-client scenarios implies coordinating the forwarding of Interests, so that all the Interests for a specific Data packet are *(i)* forwarded towards the source storing the requested Data packet, and *(ii)* forwarded on the optimal paths, such that more Interests are aggregated and the Data packets are cached in optimal points of the network. This can be achieved if the Data packets are delivered over the optimal set of multicast trees [101]. The latter implies that each node in the network needs to know where to forward each Interest, such that the Interests and the Data packets follow the computed multicast trees, which does not scale in large and dynamic networks. Moreover, the optimal set of multicast trees can only be computed by a central entity that is aware of the network topology, which is both computationally hard and difficult to implement in large dynamic networks.

An alternative solution to the use of an optimal set of multicast trees is to use network coding [7]. When network coding is enabled, the nodes perform coding operations on the received Data packets (*i.e.*, combine the Data packets), instead of just forwarding them, as in traditional networks. The network coded Data packets contain information from all the Data packets that have been combined to generate them. This reduces the granularity of the information and subsequently of the content requests. As a result, clients do not need to request specific Data packets, but rather network coded Data packets. Therefore, the nodes do not need to coordinate the faces where they forward Interests, which enables efficient multipath communication without explicit coordination mechanisms and enhances network bandwidth utilization.

In the following three examples, we illustrate the benefits that the introduction of network coding into NDN brings to multipath content retrieval.

- *Multi-source single-client* — Let us consider the case illustrated in Fig. 1.2, where

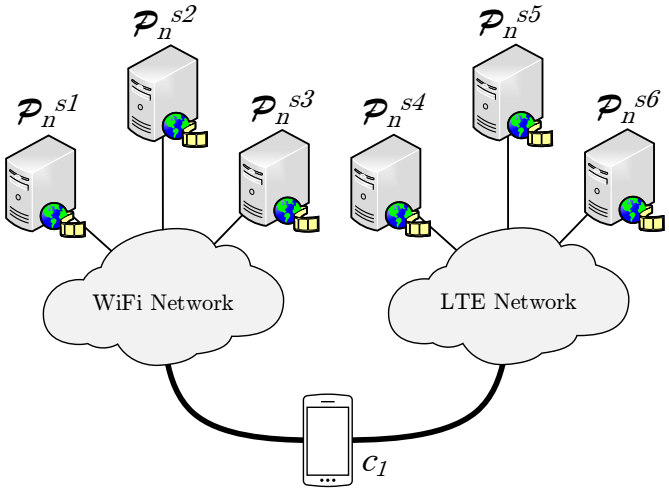

Figure 1.2: Devices retrieving Data packets over LTE and Wi-Fi: multi-source single client.

a client is interested in a content object composed of a set of Data packets. Let us also consider that the Data packets that compose the content object are distributed across multiple sources, such that each source stores a subset of Data packets. In this case, the client and the routers need to select properly the face over which they forward each Interest, so that it reaches the source that stores the requested Data packet. This can be accomplished by carefully configuring the NDN forwarding tables, which store information about the faces that a node could use to forward Interests with a particular name prefix, of all the nodes, such that each Interest reaches the right source. However, for content objects comprised of a large number of Data packets, *i.e.*, content objects for data intensive applications, keeping the forwarding tables of all the nodes updated for each Data packet does not scale well. Specifically, in large networks and in the presence of unreliable sources that can become available or unavailable at any moment, keeping the forwarding tables updated may require a lot of signaling messages, and thus waste network resources. Differently to the original NDN, in a network coding enabled NDN architecture, the clients and the routers do not need to know which sources store each Data packet and over which interface they can locate it, since the Interests are for network coded Data packets that are stored at any source rather than for specific Data packets that are stored at specific sources. This implies that the forwarding tables can be smaller than those of original NDN. Specifically, only one entry for the name prefix of the content object is needed in network coding enabled NDN, while

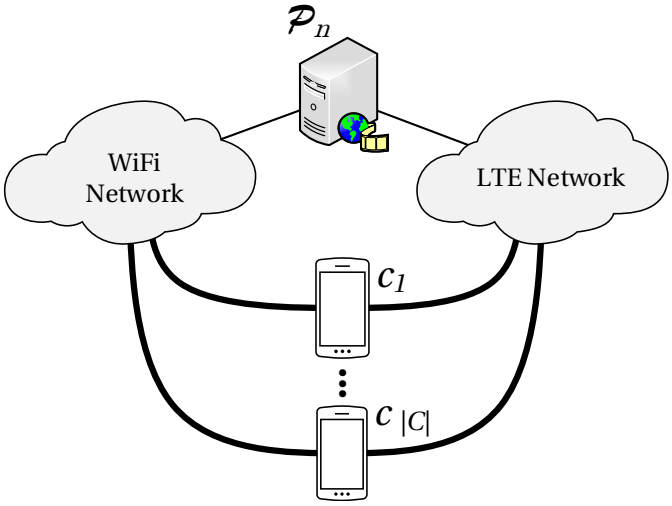

Figure 1.3: Devices retrieving Data packets over LTE and Wi-Fi: single-source multi-client.

in original NDN a distinct entry is required for the name prefix of every Data packet composing the content object, since they can be located in different sources.

- *Single-source multi-client* — Let us now examine the case where a single source stores the complete set of Data packets that compose a content object, and that multiple clients are interested in the content object, as illustrated in Fig. 1.3. To minimize the time needed for each client to receive the complete set of Data packets that compose the content object, while also minimizing the number of duplicated Data packet transmissions in the network, the Data packets need to travel over an optimal set of multicast trees [101], which, as discussed above, is does not scale well with the network size and dynamics. In the illustrative example shown in Fig. 1.3, clients are connected to the source through both LTE and Wi-Fi interfaces. If all the clients send the Interest for a particular Data packet over the LTE interface, then the requested Data packet will only be sent through the LTE network. However, if a fraction of the clients decides to send the same Interest over the Wi-Fi interface, the requested Data packet will also be sent from the source to the Wi-Fi network, wasting network resources. When multiple clients send Interests for the same content object in a network coding enabled NDN, they do not need to coordinate the Interests (*i.e.*, select the sequence numbers) that they send over each face. This is due to the fact that the Interests are for network coded Data packets. Thus, they can be aggregated

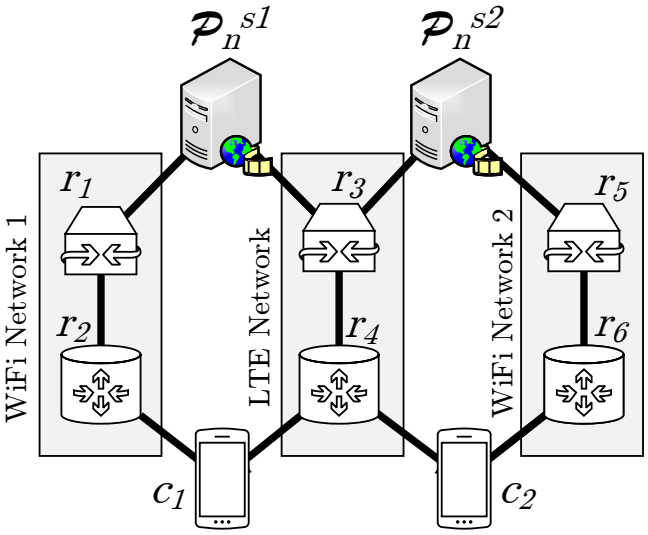

Figure 1.4: Devices retrieving Data packets over LTE and Wi-Fi: multi-source multi-client (butterfly network).

at any node, which leads to more efficient network utilization.

- *Multi-source multi-client* — Another problematic scenario is when multiple clients are interested in a content object composed of a set of Data packets that are distributed across multiple sources. In this scenario, multipath data retrieval in the original NDN suffers from the shortcomings of both the multi-source single-client and the single-source multi-client scenarios that we previously discussed. To illustrate this, let us consider the network in Fig. 1.4. In this network, two clients need to coordinate where to send each Interest, such that the LTE access router r_4 is able to aggregate the Interests for the same Data packet. Moreover, when each of the sources has a disjoint set of Data packets, the clients also need to know which Data packets each source stores, to avoid sending Interests to the source that does not store a copy of the requested Data packet. With network coding enabled, no coordination is needed at the clients nor at the routers. This is because the Interests can be satisfied by any network coded Data packet.

1.3.2 Throughput Gains in Case of Network Bottlenecks

In addition to the throughput gains due to the use of optimal multipath content retrieval, enabling network coding in NDN can alleviate throughput degradation when

bottlenecks are present in the network. In order to illustrate how the throughput is degraded by network bottlenecks, let us consider the widely known butterfly network topology shown in Fig. 1.4. In this topology, both clients c_1 and c_2 are interested in a content object composed of two Data packets which are distributed across both sources, with source s_1 storing a copy of the Data packet p_1 and source s_2 storing a copy of the Data packet p_2. It is clear that the clients c_1 and c_2 can retrieve the Data packets p_2 and p_1, respectively, only over the LTE interface. In this case, if network coding is not enabled, the LTE access node r_4 cannot aggregate the Interests sent by the clients c_1 and c_2, since they are for different Data packets. This means that the link between the nodes r_3 and r_4 becomes a bottleneck for the Data packets p_2 and p_1 traveling to the clients c_1 and c_2, respectively. Thus, one of the clients will see a higher delay in receiving the complete set of packets, which is critical for time-constrained applications such as video streaming. In contrast, in a network coding enabled NDN architecture, the node r_4 is able to aggregate the Interests sent by the clients c_1 and c_2, as these Interests are for any network coded Data packet. If the node r_3 applies network coding to the Data packets received from the sources, the resulting network coded Data packet will contain information from both sources and thus will be useful for both clients c_1 and c_2. In content intensive applications, where the content objects are formed by a large number of Data packets, the delay reduction that network coding brings to the clients accumulates over all the Data packets, bringing noticeable gains to the client's throughput.

1.3.3 Resilience to Packet Losses

Furthermore, we illustrate how the integration of network coding in the NDN architecture improves the resilience to Interest and Data packet losses. This property has been widely studied in state-of-the-art host centric content delivery scenarios [82, 88]. Similarly to traditional content delivery architectures, in NDN, network coding can deal efficiently with packet losses by eliminating the need for explicit packet retransmissions. To illustrate this, let us consider an NDN network with Interest and Data packet losses. Let us also consider that a client is interested in a content object composed of a set of Data packets, and that one of the Interests sent by the client or one of the Data packets sent to the client is lost in the network. Hence, one of the Data packets that the client is expecting will not arrive. If network coding is not enabled, the client should wait until one of the Interests that it has sent expires before realizing which Data packet will not arrive. Then, the node should retransmit the same Interest and

wait again for the Data packet. In contrast, if network coding is enabled, a proactive client that knows the average packet loss rate can send a few additional Interests, where the number of additional Interests sent depends on the packet loss rate and the finite field where the coding operations are performed. This is possible because, in the event of an Interest or Data packet loss, any other network coded Data packet will be useful to the client, even if it is not an exact copy of the missing Data packet. This reduces the time that the client needs to retrieve the complete set of Data packets, since it does not need to wait for Interest expiration to send new Interest that will compensate for the lost packets.

1.4 Thesis Objective

Motivated by the predominance of data intensive applications on the Internet, and by the promising results presented in previous attempts to integrate network coding into the NDN architecture [59, 100], the main objective of this thesis is the following:

> *to investigate how data intensive applications on NDN could benefit from the use of network coding.*

To reach the main objective of this thesis, we have the following specific objectives.

First, we need an architecture that enables network coding in NDN and that supports data intensive applications. However, the literature lacks a well defined practical architecture that integrates network coding into NDN. Previous schemes integrating network coding into NDN [59, 100] were not complete, only providing brief descriptions of simple architectures that consider specific scenarios. Thus, this thesis has the following specific objective:

> *to design, implement and evaluate a complete architecture that integrates network coding into NDN and that supports data intensive applications.*

Second, we need to choose a data intensive application that will be the subject of our study to evaluate the benefits of network coding in NDN. Since video streaming is the most important data intensive application nowadays [21], the second specific objective of this thesis is the following:

to evaluate the benefits that network coding bring to video streaming applications over NDN, for end-users and video content providers.

Third, since the default behavior of intermediate routers in NDN is to cache all the Data packets that they receive, the caches in the intermediate nodes should be extremely large in order to store the high amount of Data packets that data intensive applications require. This is unfeasible in real-world scenarios, and for that reason the third specific objective of this thesis is the following:

to design and evaluate a caching policy that takes into account the particularities of network coding enabled NDN architectures, and maintains the benefits that network coding brings to data intensive applications over NDN.

1.5 Thesis Contributions

As mentioned in the previous Section, one of the objectives of this thesis is to design a complete architecture that integrates network coding into NDN. Thus, the first contribution of this thesis is the *NetCodNDN* architecture, which enables network coding for data intensive applications in NDN. The design, implementation and evaluation of the NetCodNDN architecture is provided in Chapter 3.

Based on the proposed NetCodNDN architecture, the next contribution of this thesis is the study of the benefits that network coding brings to data intensive applications in NDN. Since video streaming is the most popular data intensive application [21], this thesis studies video streaming over network coding enabled NDN, in particular, dynamic adaptive video streaming. Hence, this thesis provides a Dynamic Adaptive Streaming architecture for NetCodNDN, called *DAS-NetCodNDN*. The details of the DAS-NetCodNDN architecture and the study of the benefits that network coding brings to video streaming applications over NDN, for both end-users and video content providers, are presented in Chapter 4.

The two previous contributions of this thesis, namely NetCodNDN and DAS-NetCodNDN, consider a scenario in which nodes have sufficient storage space to cache all the Data packets that they receive, which is infeasible in real-life. For this reason, the final contribution of this thesis is a caching policy called *PopNetCod* that

complements the design of the NetCodNDN and DAS-NetCodNDN architectures. The PopNetCod caching policy uses content popularity information to decide which network coded Data packets routers store in their caches, in order to maintain the benefits that network coding brings to data intensive applications, even when the routers have limited caching capacity.

In the following subsections, these contributions are further explained.

1.5.1 NetCodNDN: A Network Coding Enabled NDN Architecture

Motivated by the improvements that network coding brings to content retrieval over NDN, presented in Section 1.3, and in order to study the benefits that network coding brings to NDN data intensive applications, the first contribution of this thesis is *NetCodNDN*, an architecture that enables network coding in NDN. The NetCodNDN architecture is based on the NDN architecture [38, 104], with a redesigned PIT table, and new procedures for Interest and Data packet processing.

Our proposed architecture solves the shortcomings of previous proposals to enable network coding in NDN [59, 100]. Specifically, previous works propose that each Interest carries information about the Data packets retrieved by the client that sent the Interest. Then, nodes holding Data packets that match the name prefix of the Interest reply only if they can provide a network coded Data packet that is innovative to the client, based on the information added to the Interest. However, in the presence of multiple clients, *(i)* the aggregation of Interests is problematic, since Interests for the same Data packet but from different clients contain different information about the Data packets retrieved by each client, and it is not clear how this information could be aggregated for different clients; and *(ii)*, when a client sends multiple Interests in parallel to receive multiple different Data packets, it includes the same information about the Data packets it already has retrieved. This is undesirable, as a node that has a matching Data packet will reply to these Interests with the same Data packet, which will be duplicated for the client.

The NetCodNDN architecture *(i)* eliminates the need to include in the Interests the information about the Data packets available at the client, thus, simplifying Interest aggregation; *(ii)* allows the network nodes to keep information about the Data packets they have sent on each face, reducing the number of duplicate Data packets; and *(iii)* enables clients to send multiple concurrent Interests for different Data packets of the

same content object, by modifying the way in which the nodes process the Interest messages.

The NetCodNDN architecture has been implemented using open-source and well-known libraries, and follows the NDN project guidelines to simplify its re-usability and expansion. In particular, it uses the NDN Forwarding Daemon (NFD) codebase [61] as a base for the NetCodNDN architecture. To enable network coding, it uses Kodo [69], an industry standard C++ library that implements network coding functionality in an optimized way.

The experimental evaluation shows that NetCodNDN leads to significant improvements in terms of content retrieval delay compared to the original NDN. Our results demonstrate that NetCodNDN improves multipath content retrieval which offers large gains in terms of the time needed to retrieve content objects. This translates into increased throughput seen by the clients. Moreover, it permits to exploit more efficiently the available network resources in multi-source and multi-client scenarios, even when bottlenecks are present in the network. Finally, it also shows increased robustness to packet losses.

The research work related to this contribution has resulted in the following publication:

> Jonnahtan Saltarin, Eirina Bourtsoulatze, Nikolaos Thomos, and Torsten Braun. "NetCodCCN: A Network Coding Approach for Content-Centric Networks". In: *Proc. of IEEE INFOCOM'16*. San Francisco, USA, Apr. 2016. doi:10.1109/INFOCOM.2016.7524382

1.5.2 Adaptive Video Streaming over Network Coding Enabled NDN

The next contribution of this thesis is an architecture that enables adaptive video streaming over NetCodNDN. As of 2015, video accounted for 70% of consumer Internet traffic, and it is expected to reach 82% by 2020 [21]. This increase in the volume of video traffic is fueled by the emergence of applications such as social video, virtual reality (VR), augmented reality (AR), etc., that have become popular and involve the delivery of large amounts of video content. For this reason, this thesis focuses on the study of the benefits that network coding brings to video streaming over NDN, for both end-users and video content providers.

One of the most popular and efficient approaches available for video streaming is Dynamic Adaptive Streaming over HTTP (DASH) [37, 85]. The advantage of DASH compared to previous video streaming methods is that the clients are in control of the streaming logic. Thus, the clients are able to decide the appropriate bitrate and resolution of the requested video, among the multiple options offered by the video content providers. This requires that the sources encode the video in different representations, *i.e.*, different bitrates and resolutions. Each representation is further divided into a series of video segments with a duration of a few seconds. This allows clients to adapt in real-time the demanded video resolution and bitrate in response to changes in the network conditions.

The client driven video adaptation property has ranked DASH as an attractive option for video streaming in future Internet architectures [99], in particular for NDN. The content retrieval mechanism of NDN resembles the video retrieval mechanism of DASH [46]. In particular, NDN and DASH are both client driven. Hence, clients request content by sending requests with the names assigned to each piece of content, *i.e.*, a content object in NDN or a video segment in DASH. It is, therefore, natural to consider DASH for implementing adaptive streaming in the NetCodNDN architecture.

Thus, to study the benefits that network coding bring to video streaming over NDN, this thesis proposes a Dynamic Adaptive Streaming over NetCodNDN (DAS-NetCodNDN) architecture. In comparison to previous works proposing dynamic adaptive streaming over NDN [25, 45, 46, 73, 98], DAS-NetCodNDN exploits network coding to efficiently use the multiple paths connecting the clients to the sources. Moreover, DAS-NetCodNDN enables efficient multi-source video streaming and improves resiliency to Data packet losses. The performance gains are verified through simulations in a Netflix-like scenario. The evaluation shows that video streaming applications over NDN benefit from the use of network coding in the form of reduced load on the video sources, increased cache-hit rate at the in-network caches and improved video quality at the clients.

The research work related to this contribution has resulted in the following publication:

> Jonnahtan Saltarin, Eirina Bourtsoulatze, Nikolaos Thomos, and Torsten Braun. “Adaptive Video Streaming with Network Coding Enabled Named Data Networking”. In: *IEEE Transactions of Multimedia*, vol. 19, no. 10,

Aug. 2017. doi:10.1109/TMM.2017.2737950

1.5.3 Caching Policy for Network Coding Enabled NDN

The study of the benefits that network coding brings to data intensive applications presented in Sections 1.5.1 and 1.5.2 assumes that the intermediate routers have enough caching capacity to store all the Data packets that are transmitted through them. This allows to focus on studying the benefits that network coding brings to NDN data intensive applications, without the need to take into consideration the impact that the different caching policies may have on the proposed architectures. However, assuming unlimited caching capacity for the intermediate routers is not realistic.

To study the benefits that network coding brings to data intensive applications in the presence of intermediate routers with limited caching capacity, we enhance the design of the NetCodNDN architecture and propose PopNetCod, a popularity based caching policy for NetCodNDN. In PopNetCod the intermediate routers distributedly estimate the popularity of the content objects based on the received Interests. Then, based on this information, each node decides which Data packets it inserts into or evicts from its cache. The decision to cache a particular Data packet is taken before the Data packet arrives at the node, while processing the corresponding Interest. Since the first nodes to process Interests in their path to the source are the edge nodes, this helps to cache the most popular Data packets closer to the network edges, which reduces the content delivery delay [24, 26, 87]. To avoid that two independent nodes cache the same Data packet, when a node decides to cache the Data packet that is expected as a reply to a received Interest, it informs the nodes upstream in the path by setting a binary flag in the Interest. This increases the diversity of network coded Data packets in the caches. When the cache of a node is full and a Data packet should be cached, the node decides the name prefix and number of Data packets that should be evicted from its cache based on the popularity information. Since network coded Data packets with a specific name prefix are equivalent in terms of contained information, the node does not need to decide which particular Data packets to evict, and can choose random Data packets with the selected name prefix.

The research work related to this contribution has resulted in the following manuscript, currently submitted for publication:

Jonnahtan Saltarin, Torsten Braun, Eirina Bourtsoulatze, and Nikolaos Thomos. "PopNetCod: A Popularity-based Caching Policy for Network Coding enabled Named Data Networks". *Submitted for publication.*

1.6 Thesis Outline

The rest of this thesis is organized as follows.

Chapter 2 provides a background and overview of the existing works in the area of content retrieval, Named Data Networking, information coding, dynamic adaptive video streaming, and caching algorithms, which are the base of this thesis.

Chapter 3 presents NetCodNDN, an architecture for enabling network coded content retrieval over Named Data Networking.

Chapter 4 presents a Dynamic Adaptive Streaming over NetCodNDN (DAS-NetCodNDN) architecture. Moreover, Chapter 4 presents the results of the study of the benefits that network coding brings to video streaming over NetCodNDN.

Chapter 5 presents PopNetCod, a popularity based caching policy that complements the design of the NetCodNDN architecture.

Chapter 6 concludes this thesis and presents future directions for network coding enabled NDN architectures.

2

State of the Art

2.1 Overview

Retrieving content (*e.g.*, files, video streams, personal messages, *etc.*) over the Internet is intensely growing and expected to continue growing in the near future [21]. Moreover, the way in which users retrieve content over the Internet is changing, from connecting to a particular network node to obtain the content that it publishes, to requesting particular content objects, without caring where the content is being served from. This growth in content transmission and the change in user behavior have prompted the development of new paradigms for future Internet architectures. Information-Centric Networking (ICN) is a novel Internet paradigm that gives more importance to the characteristics of the content than to its location in the network. Named Data Networking (NDN) is one of the multiple ICN architectures that have been proposed, and which has became very popular due to its large research community and its open-source implementations. However, the NDN architecture does not achieve optimal content retrieval in multi-client and multi-source scenarios in which bandwidth is limited, bottlenecks exist in the network, and packet losses occur.

One approach to alleviate these issues is to enable information coding in the network. Fountain coding is a popular class of information coding in which the source is able to generate a large number of coded Data packets, and the client only needs to retrieve a subset of those coded packets to decode the original content. However, it has been demonstrated that by allowing the intermediate nodes to apply coding operations, the performance of content retrieval increases. For this reason, network coding has been proposed as an alternative information coding approach. The benefits of network coding have been demonstrated in traditional Internet architectures, and also a few studies demonstrate that they could be beneficial to the NDN architecture.

In this Chapter, we provide a detailed description of the concepts that we will use during this thesis, and an analysis of the most relevant related works. We start by describing the process of content retrieval over the Internet. Then, we introduce ICN and its most popular realizations. Among those, in this thesis we use the NDN architecture, and thus we provide a detailed description of it, including its data structures and packet processing procedures. Next, we discuss the most relevant information coding methods that are used to solve the issues of the NDN architecture presented in Chapter 1. Then, we focus on network coding, the information coding method that we choose to use in this thesis, and discuss some previous attempts to integrate network coding into the NDN architecture. Then, since nowadays video streaming is one of the most popular Internet applications, we present some concepts of video streaming, in particular dynamic adaptive video streaming and its integration into the NDN architecture. Finally, since one of the most important features of the NDN architecture is in-network caching, we discuss some of the most important caching algorithms for NDN.

2.2 Content Retrieval over the Internet

Content retrieval over the Internet consists in transmitting a copy of a certain content object from a content provider to a set of end-users that have expressed interest in the content object. The end-users can use different methods to express interest in one or more content objects. For example, in the current Internet, the end-users can send `HTTP GET` [27] requests to the network address of the server containing the content, or they can subscribe to a `PUSH` [92] service, among other methods.

A content retrieval scenario consists of a set of sources $\mathscr{S}$ that are associated with

content providers and that serve content objects, a set of clients $\mathscr{C}$ that are associated with end-users and that request content objects, and a set of routers $\mathscr{R}$ through which the content requests and content objects are transmitted. In this thesis, we consider that sources are network nodes that reply to content requests with the content data.

Content retrieval in the current Internet, based on the IP protocol [18], requires clients to know not only *what* content they desire, but also *where* the desired content is located in the network, *i.e.*, the IP address of the source that holds a copy of the requested content. This requires a system that permits clients to resolve a name into an IP address, *e.g.*, the Domain Name System (DNS) [57, 58]. In turn, the sources also need to know the IP address of the client to which they should send each requested content object. The routers in the network forward the content requests and content objects based on the IP address of the destination node.

However, content retrieval in the current Internet is not aligned with the behavior of the Internet users nowadays: users care more about the characteristics of the content they want to obtain rather than where this content is located in the network. To address the mismatch between the Internet users' behavior and the Internet architecture, new communication paradigms have been proposed in an attempt to define a future Internet architecture. In particular, Information-Centric Networking (ICN) is a new communication paradigm in which the importance is shifted from *where* the content is located, to *what* the content is. In ICN, each piece of content has a *name* associated to it, which is used to route the content through the network. In this way, content becomes independent from location, application, storage, and means of transportation. The benefits of such content-centric approach are improved content retrieval efficiency, better scalability with respect to information/bandwidth demand and higher robustness in challenging communication scenarios [71]. In the following Section, we provide details of multiple ICN proposals available in the literature.

2.3 Information-Centric Networking

Multiple ICN based Internet architectures have been proposed in the literature, such as the Data-Oriented Network Architecture (DONA) [42], the Publish-Subscribe Internet Routing Paradigm (PSIRP) [29, 94], the Network of Information (NetInf) [6] architecture, and the original Content-Centric Networking (CCN) [38] and its successor Named Data Networking (NDN) [104]. Next, we give a brief description of

the aforementioned ICN architectures. For a more comprehensive description of the available ICN proposals, the interested reader is referred to [5] and [102] for complete surveys.

2.3.1 Data-Oriented Network Architecture

In the Data-Oriented Network Architecture (DONA) [42], content objects are published by *principals*, and named by a combination of the principal ID and the object ID given by the principal. The principals register the content name in their local *Resource Handler*, which are content routers. Then, the registration message is propagated in the network, such that other resource handlers know where to find that particular content object. Clients interested in a certain content object send a *Find* message to its associated resource handler, which forwards the request towards the appropriate resource handler. The content object is delivered to the client following the same route as the Find message, or an alternative route. Resource handlers can cache content objects as they are forwarded to the clients.

2.3.2 Publish-Subscribe Internet Routing Paradigm

In the Publish-Subscribe Internet Routing Paradigm (PSIRP) [29, 94] architecture, *publishers* are sources that provide content objects to the PSI network. The *subscribers* are clients that explicitly express their interest in a particular content object by issuing subscription messages. Each content object is associated with a scope, which could be a physical scope, *e.g.*, only available at the campus of the "University of Bern", or a logical scope, *e.g.*, only available for users of the social network "Facebook". Thus, the scope is used to limit the reachability of the content object to clients that have access to that particular scope. Moreover, each content object has a unique name that allows subscriber interests to be matched with the content object. The matching between subscriber interests and content objects occur in a node called Rendezvous Point (RP). The RPs are able to store subscriptions for a certain time, until the content object becomes available. This functionality allows the publication and subscription operations to be decoupled in time and space, since clients can send subscription messages for content objects that still do not exist in the network.

2.3.3 Network of Information

The Network of Information (NetInf) [6] architecture enables two types of content retrieval. In the first approach, content object requests are routed towards their source locations based on the names of the requested content objects. The content objects are forwarded from their source locations to the requesting clients also based on their name. In the second approach, a name resolution step is introduced, which provides the location of the content object sources based on the content object names. This enables a graceful transition between the current location-based Internet architecture and a purely name-based Internet architecture. Three types of messages exist in the NetInf architecture: *(i)* PUBLISH, which is used by sources to announce their available content objects, such that the requests can be routed towards them and the name resolution mechanism is able to resolve names into its correct location(s); *(ii)* GET, which is used by clients to request content objects; and *(iii)* SEARCH, which allows clients to query the network by sending a set of keywords, and receiving specific names and possibly also locations of content objects that match the query.

2.3.4 Content-Centric Networking

Content-Centric Networking (CCN) [38] is an architecture in which the content is described by a *name* and the clients demand content with the help of *Interest* messages that contain the name of the requested content. The Interests are transmitted over the network until they reach a node holding a copy of the content whose name matches that of the Interest. This node creates a *Data packet* that contains a copy of the requested content and sends it back to the client. The Data packet follows the reverse path of that followed by the Interest. As the Data packet is transmitted backwards to the client, intermediate routers can store copies of it, so they can reply to future Interests sent by other clients for the same content.

Based on the CCN architecture, the Named Data Networking (NDN) [104] architecture has been proposed as part of a large research project [105] conducted by multiple research institutions, including the authors of the CCN architecture. After a few years of common design decisions, the NDN architecture has parted away of the CCN architecture, and evolved in a different direction. For example, NDN routers perform longest prefix match to identify matching Data packets in the cache, while CCN routers perform exact matching in order to simplify the forwarding procedures.

A comprehensive list of differences between NDN and CCN could be found at [36]. Nevertheless, both architectures share the same basic principles, *e.g.*, communication is based on the content name, content is requested with Interests and delivered in Data packets.

Among the proposals described in Section 2.3, CCN and NDN are the most popular in the literature, due to their active research community and their open-source implementations. In this thesis, we use a software implementation based on the CCN architecture for the evaluation presented in Chapter 3, and a software implementation based on the NDN architecture for the evaluations presented in Chapters 4 and 5. However, the most prominent features that we use from CCN and NDN are similar in both architectures. In this thesis, we will focus on the NDN architecture, but the work presented here could be easily implemented in the CCN architecture. In the following Section, we provide a detailed description of the NDN architecture.

2.4 Named Data Networking

As presented in Section 2.2, a content retrieval scenario consists of a set of source nodes $\mathscr{S}$ that generate and store content objects, a set of clients $\mathscr{C}$ that demand content objects and a set of intermediate nodes $\mathscr{R}$ through which the content objects are requested and transmitted. Every node $v \in \mathscr{S} \cup \mathscr{C} \cup \mathscr{R}$ is connected with its neighboring nodes through a set of faces $\mathscr{F}_v$.

2.4.1 Content Object Partitioning and Naming in NDN

Since the size of content objects is usually larger than the Maximum Transmission Units (MTU), the content objects are divided into smaller Data packets that fit into an MTU. Therefore, we consider that a content object is composed of a set of Data packets $\mathscr{P}_n = \{p_{n,1}, \ldots, p_{n,|\mathscr{P}_n|}\}$, where n is the name of the content object, which serves as a name prefix for the Data packets, and $|\mathscr{P}_n|$ is the number of the Data packets that compose the content object $\mathscr{P}_n$. The name of each Data packet $p_{n,j} \in P_n$ is generated by appending the segment ID j to the name n of the content object. For instance, the name of the Data packet $p_{n,1}$ is `/provider/videos/largevideo.h264/1`, where n =`/provider/videos/largevideo.h264` is the name of the content object and j =`1` is the segment ID. To retrieve a content object composed of the set of Data packets $\mathscr{P}_n$, a client $c \in \mathscr{C}$ should send a set of Interests $\mathscr{I}_n = \{i_{n,1}, \ldots, i_{n,|\mathscr{P}_n|}\}$, where

NDN FIB	
Name	Faces
(n)	*f***
(n')	*f*,f****
(n'')	*f*,f***

Figure 2.1: Forwarding Information Base (FIB).

$|\mathscr{I}_n| = |\mathscr{P}_n|$, *i.e.*, there is one Interest for each Data packet. Note that the actual number of Interests that the client c should send to retrieve $\mathscr{P}_n$ may be higher than $|\mathscr{P}_n|$, since in a lossy network some Interests and Data packets may be lost. Thus, some of the Interests in $\mathscr{I}_n$ will need to be sent more than once. Interests have an expiration time, which is used to define the validity of the content request. When Interest expires, the client could consider that the Data packet that was supposed to arrive as a reply to this Interest will not arrive (*e.g.*, because of a packet loss), and retransmit the Interest. The clients can send multiple Interests of the set $\mathscr{I}_n$ in parallel, *i.e.*, without waiting to receive Data packets before sending new Interests, to speed-up the retrieval of the set of Data packets $\mathscr{P}_n$. This process is known as *Interest pipelining*.

2.4.2 Data Structures of the NDN Architecture

NDN nodes have three tables in which they store information used to facilitate the content retrieval process. This tables are: the *Forwarding Information Base*, where the node stores information about the faces that it can use to forward Interests for a specific name prefix; the *Content Store*, where the node caches Data packets that pass through it, and that could be usefull to reply to future Interests; and the *Pending Interest Table*, where the node keeps track of the Interests that it has forwarded and the faces over which these Interests have arrived. In the following subsections, we describe these three tables of the NDN architecture.

Forwarding Information Base

The Forwarding Information Base (FIB) is a table in which the node v stores the faces $\mathscr{F}_n^v$ over which it can forward Interests with name prefix n to retrieve the correspond-

NDN CS	
Name	Entry
$(n,1)$	$p_{n,1}$
$(n,2)$	$p_{n,2}$
$(n,3)$	$p_{n,3}$

Figure 2.2: Content Store (CS).

ing Data packets. The FIB entries are of the form $(n, \mathcal{F}_n^v)$, as depicted in Fig 2.1.

Content Store

An NDN node v maintains a cache with a set of Data packets $\mathcal{P}_n^v$ that it has received and are considered useful to store, in order to reply to future Interests for the name prefix n. Each Data packet $p_{n,j} \in \mathcal{P}_n^v$ is stored as an entry in the CS, as depicted in Fig. 2.2. When a node v receives an Interest $i_{n,j}$, it looks into its CS to find all the entries that match the name prefix (n, j) of the Interest. Since the name prefix (n, j) refers to a specific Data packet, only one entry will match.

The CS provides methods to insert new Data packets and to get Data packets that it holds. This methods are detailed below.

- `GetCS(`n, j`)` – Gets the Data packet $p_{n,j}$ that matches the name prefix (n, j), if it exists in the CS of the node v.
- `InsertCS(`$p_{n,j}$`)` – Inserts a new Data packet $p_{n,j}$ into the CS of the node v, if it is not already cached in the CS.

Pending Interest Table

An NDN node v stores the information about the Interest that were forwarded upstream in the Pending Interest Table (PIT). The PIT is a collection of entries $\mathcal{T} = \{t_{n,j} \ldots\}$. Each PIT entry $t_{n,j}$ keeps track of the received Interests with name prefix (n, j) that were forwarded and are pending, *i.e.*, have not been consumed by a Data packet. Each entry $t_{n,j}$ has two components for each face f, *(i)* an *in-record* $t_{n,j}^{f(in)}$ that

NDN PIT			
Name	Entry		
(n,1)	f^{+}	f^{++}	$f*$
	in: 1	*in: 1*	*in: 0*
	out: 0	*out: 0*	*out: 1*
(n,2)	f^{+}	$f*$	$f**$
	in: 1	*in: 0*	*in: 0*
	out: 0	*out: 1*	*out: 1*

Figure 2.3: Pending Interest Table (PIT).

takes the value "1" if an Interest with name prefix (n, j) has been received over face f and it has not been satisfied, or the value "0" otherwise, and *(ii)* an *out-record* $t_{n,j}^{f(out)}$ that takes the value "1" if an Interest with name prefix (n, j) has been sent over face f and the expected Data packet has not arrived, or the value "0" otherwise.

The NDN PIT provides methods to insert information about new Interests, and to get the PIT entry associated with a particular name prefix. In particular, the following two methods are provided:

- `GetPIT((`n, j`))` – Gets the PIT entry $t_{n,j}$ associated with the name prefix (n, j).
- `InsertInPIT((`n, j`),`f`)` – Sets to "1" the value of $t_{n,j}^{f(in)}$ in the PIT entry $t_{n,j}$ associated with the name prefix (n, j). If the PIT entry does not exist, it is created by this method.
- `RemoveInPIT((`n, j`),`f`)` – Sets to "0"the value of $t_{n,j}^{f(in)}$ in the PIT entry $t_{n,j}$ associated with the name prefix (n, j). If the PIT entry does not exist, this method does nothing.
- `InsertOutPIT((`n, j`),`f`)` – Sets to "1" the value of $t_{n,j}^{f(out)}$ in the PIT entry $t_{n,j}$ associated with the name prefix (n, j). If the PIT entry does not exist, it is created by this method.
- `RemoveOutPIT((`n, j`),`f`)` – Sets to "0" the value of $t_{n,j}^{f(out)}$ in the PIT entry $t_{n,j}$ associated with the name prefix (n, j). If the PIT entry does not exist, this method does nothing.

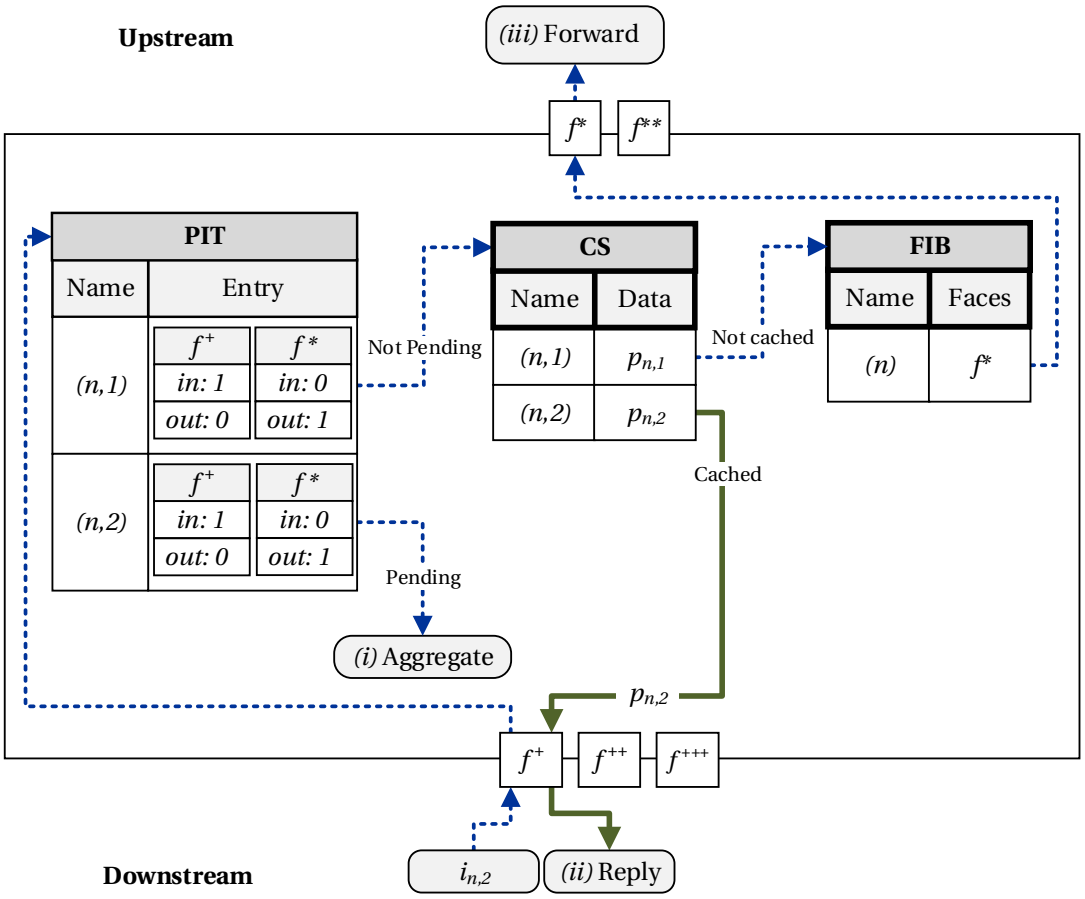

Figure 2.4: Interest processing in NDN.

2.4.3 Interest Processing

In NDN, when a node receives an Interest, it either: *(i)* adds information about the received Interest to the PIT and waits for the requested Data packet to arrive, if previously an Interest for the same Data packet has already been forwarded; *(ii)* replies to the Interest with a matching Data packet from its CS; or *(iii)* forwards the Interest to neighboring nodes, in order to receive the requested Data packet. This procedure is depicted in Fig. 2.4 and it is further explained below.

(i) Waiting for a new Data packet — Consider a node v that receives an Interest $i_{n,j}$ over face f. If the node v finds in its PIT an entry that matches the name in the Interest, it means that it has already forwarded $i_{n,j}$ and hence the Data packet $p_{n,j}$ is expected. In this case, the node adds face f over which the Interest has arrived to the respective PIT entry, and does not forward $i_{n,j}$ again. In this way,

the node reduces the number of Interests in the network, and the number of duplicate Data packets that it receives. This procedure is also known as *Interest aggregation.*

(ii) Replying to an Interest — When the PIT of the node v does not have any entry that matches the Interest $i_{n,j}$, it looks for a Data packet with a matching name in its CS. If a copy of the Data packet $p_{n,j}$ is cached in the CS, the node v replies to the Interest $i_{n,j}$.

(iii) Forwarding an Interest — If the CS of the node v is not caching a Data packet that matches the name of the Interest $i_{n,j}$, the node v forwards the Interest to one or more of its neighboring nodes, according to its FIB. Moreover, the node v also updates its PIT table to add the information about the forwarded Interests.

2.4.4 Data Message Processing

Once the requested Data packet $p_{n,j}$ is found in the CS of a router or in a source, it is sent to the client over the reverse path of that followed by the Interest. When a node receives a Data packet $p_{n,j}$ over a face f, it first looks up in its PIT for an entry that matches the name of the Data packet $p_{n,j}$. If no PIT entry matches the name (n, j), the Data packet is considered unsolicited and it is discarded. If a PIT entry matches the name (n, j), the Data packet is forwarded over all the faces specified in the corresponding PIT entry. Additionally, the Data packet $p_{n,j}$ may be added to the CS, according to the caching policy of the node. A more detailed description of the NDN operation is provided in the NFD Developer Guide [4].

2.4.5 Multipath Data Delivery in NDN

As described in Chapter 1, a possible solution to the lack of sufficient bandwidth in the links connecting the network nodes is the use of multiple paths to retrieve content. The NDN architecture provides inherent support for multipath content retrieval, by allowing clients to distribute the set of Interests $\mathscr{I}_n$ needed to retrieve a content object over all their available faces $\mathscr{F}_c$, without the need to set-up multiple connections, as in multipath TCP (MP-TCP) [28]. The use of multipath communication in NDN has been studied by previous works [33, 77, 81]. These works mainly focus on the design of Interest forwarding strategies that exploit all the available interfaces of the node to distribute the Interests, without considering a specific application. Rossini

et al. [77] investigated multipath Interest forwarding strategies, showing that the use of multiple paths simultaneously can increase resilience to network dynamics and reduce repository load. They also show that naïve multipath Interest forwarding strategies (*e.g.*, round robin) could reduce the caching efficiency since when multiple clients decide to send the Interest $i_{n,j}$ to retrieve the Data packet $p_{n,j}$ over different paths, all the caches over the followed paths will cache a copy of the same Data packet $p_{n,j}$. Schneider *et al.* [81] presented novel Interest forwarding strategies that improve end-user Quality of Experience (QoE) and reduce the clients' access cost and power consumption. To accomplish this, a new set of interface estimators are proposed, which enable fine-grained control and selection of interfaces in a multi-homed scenario. However, it is not clear how the QoE is affected when multiple multi-homed clients request the same content. Gomes *et al.* [33] presented a load balancing strategy that enables simultaneous utilization of the LTE and Wi-Fi interfaces at the clients to retrieve Data packets. The proposed strategy takes into account the real-time interface conditions, *i.e.*, transmission rate, round-trip time, and packet loss rate, in order to reduce energy consumption and increase throughput at the clients, while reducing the costs for the operators. The evaluation results showed that by using the proposed strategy, network resources are more efficiently used. Moreover, clients are able to retrieve the desired content faster than by using only LTE, or using Wi-Fi offloading strategies. However, when multiple clients request a particular content object, each client still needs to coordinate with the other clients the face over which it sends each Interest, such that the network resources are used optimally.

In this thesis, we argue that optimal multipath content retrieval in multi-client and multi-source NDN scenarios could be achieved by using network coding [7]. This is possible because the sources send different network coded Data packets over each path, increasing the diversity of Data packets that are cached in the network. In this way, clients do not need to agree where to send each Interest, such that they travel over the optimal paths in the network.

Before introducing network coding in Section 2.6, in the following Section we introduce fountain codes, one of the most popular coding approaches used in content retrieval.

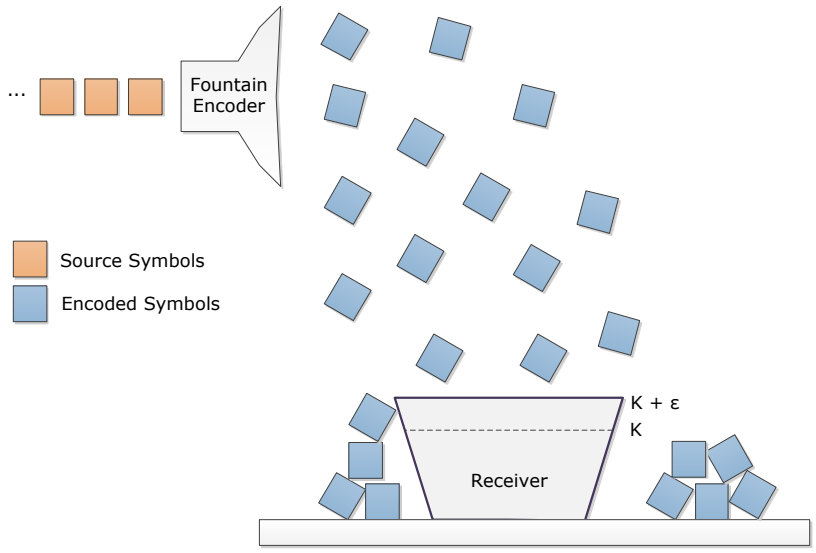

Figure 2.5: A metaphorical example of fountain codes.

2.5 Fountain Codes

Fountain codes [15] are a class of channel codes with the property that a very large number of encoded packets can be generated at the sources by randomly combining a finite set of source packets by means of XOR operations. This property simplifies the adaptation to varying loss rates.

Fountain codes can be seen as a source that generates a stream of encoded packets until the client is able to decode the K original packets. The number of packets that are combined to generate an encoded packet is determined by a degree distribution function. The clients can decode the K source packets with high probability after retrieving a subset of encoded packets with size equal to or slightly larger than the number of source packets. When the client is able to decode, it sends a signal to the source to stop the transmission. This should occur after generating $K+\epsilon$ packets, where ϵ is the number of extra encoded packets that should be transmitted. In a system without decoding feedback from the clients, the value of ϵ is decided in advance so that the probability to decode a set of packets is close to 1 when the client receives $K+\epsilon$ coded packets. As it is shown in Fig. 2.5, the encoder can be thought of as a metaphorical fountain and the output packets as drops of water. Once the client, a metaphorical bucket, has collected $K+\epsilon$ encoded packets (*i.e.*, the bucket is filled), the entire message can be decoded.

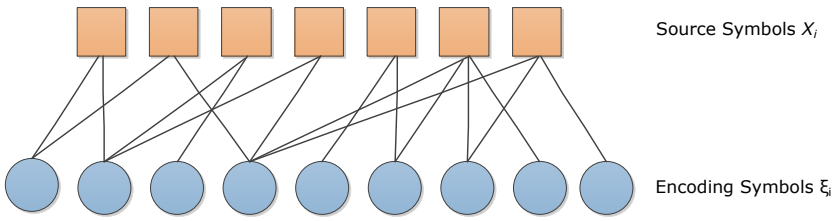

Figure 2.6: Tanner graph produced by LT encoding.

Since the encoded packets are created by randomly combining the source packets, a small header called Encoding Symbol ID (ESI) is appended to each packet, to inform the client about the source packets that were used to generate the encoded packet. The ESI is the seed of the pseudorandom generator used for generating the encoded packet (degree and source packets to be combined). The size of the ESI depends on the number of source packets, *e.g.*, for 64K source packets, an ESI header of two bytes is sufficient.

An interesting property of fountain codes that could be exploited in multi-source scenarios is that the same K original packets can be encoded at all the sources and, as the number of encoded packets that can be generated is much larger than K, each source will generate encoded packets that differ with high probability from the ones generated by other sources.

Thus, fountain codes can be used to improve the data retrieval process in presence of multiple sources, multiple paths and lossy communication channels. In the following Sections we describe two of the most popular practical implementations of fountain codes. Detailed surveys of fountain codes are given in [56, 91].

2.5.1 LT codes

Luby Transform (LT) codes [50] were the first practical realization of fountain codes. Their encoding consists in random combinations of the source packets. Each encoded packet $\hat{p}_{n,j}$ is represented with a vector $\gamma_{n,j}$ containing ones at the positions corresponding to the indices of the source packets combined. All the encoded packets define the generator matrix $\Gamma = [\gamma_{n,1}\gamma_{n,2}\cdots\gamma_{n,K}]$ where K is the number of generated packets.

The matrix Γ can be represented by a Tanner graph, as shown in Fig. 2.6, in which the

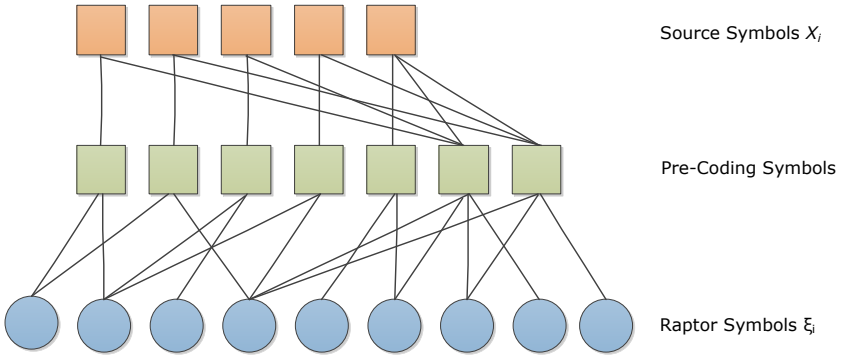

Figure 2.7: Encoding of non-systematic Raptor codes.

packets are referred to as nodes and the edges between source and encoded nodes are generated taking Γ as the connectivity matrix of the graph. An encoded packet $\hat{p}_{n,j}$ is said to be covered if there exists an edge between this packet and at least one source packet.

After transmitting these packets over a lossy network, each client is able to form a modified generator matrix Γ' in which the lost packets have been pruned from Γ. After collecting enough encoded packets to make Γ' full rank, the client can decode the original source packets using either the Belief Propagation algorithm [68] or by Gaussian elimination.

2.5.2 Raptor Codes

Raptor codes are an extension to LT codes proposed by Shokrollahi [83] in order to minimize the overhead (extra packets) needed to decode the encoded source packets.. They are designed to achieve linear encoding-decoding time complexity. The overhead of Raptor codes tends asymptotically to zero for very large codeblocks (*i.e.*, content objects), while for short codeblocks, the original content can be recovered with high probability with a set of packets slightly larger than the set of source packets, *i.e.*, one or two extra packets are enough.

The encoding procedure consists of two steps: first, the data is pre-coded with codes like Low Density Parity Check (LDPC) codes [31], Low Density Generator Matrix (LDGM) codes [76], or Tornado codes [52]. Then, the pre-coded data is encoded with weakened LT codes [50], *i.e.*, LT codes with very sparse parity-check matrices. Due

to the sparse LT codes generator matrices, linear encoding and decoding times are achieved. The pre-coding step makes the decoding procedure more robust to erasures, as the errors remaining after LT decoding can be corrected by the pre-coder. Fig. 2.7 shows a general encoding graph for Raptor codes.

An efficient implementation of Raptor codes has been presented as a 3GPP standard [1]. In this implementation, the pre-coding step consists of regular LDPC codes followed by high density Half codes. The 3GPP version of Raptor codes [1] provides a fast algorithm for solving the equations systems for packet decoding, by employing a variant of classical Gaussian elimination process.

A more recent extension of Raptor codes is RaptorQ codes [51], which currently are the closest solution to an ideal fountain code, *i.e.*, with high probability, the clients are able to decode the K source packets after retrieving K coded packets. It has the capability of achieving constant per-symbol encoding/decoding cost, and provides superior support for larger codeblocks.

2.5.3 Fountain Codes in NDN

As discussed in the previous sections, the use of fountain codes alleviates the performance drops in case of Data packet losses. Moreover, in architectures where caches exist in intermediate routers, like NDN, the use of fountain codes brings additional benefits. In particular, since the number of encoded Data packets that can be generated is very high compared to the number of original Data packets, the diversity of Data packets cached in intermediate nodes could be drastically increased, which reduces the probability of duplicated information being cached at the intermediate routers. For this reason, Parisis *et al.* [67] studied the benefits that fountain codes in combination with in-network caching, bring to multi-source NDN scenarios. The simulations show that the use of fountain codes reduces the number of duplicate Data packets in multi-source scenarios, increasing the overall performance of the network. Anastasiades *et al.* [8] proposed RC-NDN, a Raptor coding enabled NDN architecture. Evaluations show that RC-NDN outperforms the original NDN significantly. RC-NDN is particularly efficient in dense environments, where retrieval times can be reduced by 83% and the number of Data transmissions by 84.5% compared to NDN. However, the evaluation of RC-NDN only considered single-hop communication. The performance gains may be drastically reduced in multi-hop communication, in which case network coding may be required to keep the performance gains. In the next section,

Figure 2.8: Information coding at each node.

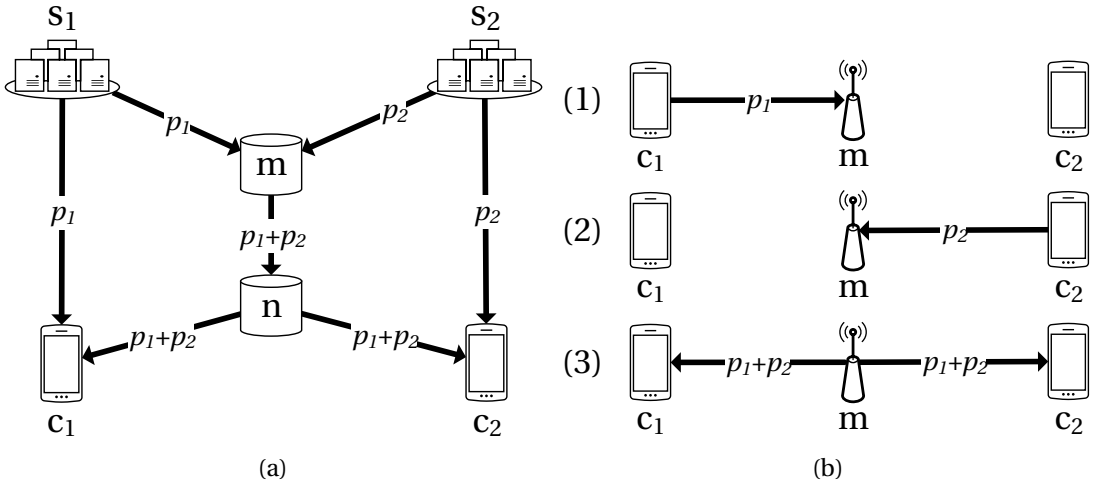

Figure 2.9: Wireline (a) and wireless (b) examples of network coding advantages.

we provide more details about network coding and its use in the NDN architecture.

2.6 Network Coding

Network coding has emerged as a paradigm of research in information theory and data networking. The idea of network coding, first presented by Ahlswede *et al.* [7], is to allow the network nodes to perform information coding, besides forwarding and replication operations like in traditional systems. By information coding we mean a mapping from inputs to outputs, as shown in Fig. 2.8. Information flows can therefore be "combined" during the course of routing. In contrast to previous coding methods, the encoding and decoding operations are not restricted to the terminal nodes (*i.e.*, clients and sources), and may happen at all nodes across the network. Allowing intermediate routers to network code the Data packets permits to avoid the conservative end-to-end policies of traditional schemes, like Forward Error Correction (FEC), and increase the Data packet diversity in multipath scenarios. The throughput gains of network coding can be illustrated with two classic examples in the literature, one for the wireline setting and one for the wireless setting, as shown in Fig. 2.9

The most famous example of the benefit of Network Coding was given by Ahlswede *et al.* [7], who considered the problem of multicast in a wireline network. Their exemplary network, commonly referred to as the butterfly network, shown in Fig. 2.9(a), represents a multicast scenario where two sources s_1 and s_2 transmit two packets p_1 and p_2 to the clients c_1 and c_2. All the links have unit capacity of one packet per timeslot. If network coding is not allowed, the node m would receive packets p_1 and p_2 in the first timeslot and would take two timeslots to transmit them through the bottleneck edge $m->n$, since this link can accommodate either packet p_1 or packet p_2, but not both at the same time. At the end, one of the two clients should wait four timeslots (when m has a cache) instead of three to complete downloading, which may affect time-constrained applications. Instead, if network coding is allowed, the node m could encode packets p_1 and p_2 and transmit only the combination $p_1 \oplus p_2$ of the two packets through the bottleneck. Then, at the third timeslot, the client c_1 has packets p_1 and $p_1 \oplus p_2$, and implementing $p_1 \oplus (p_1 \oplus p_2)$ it can recover p_2. The same occurs in client c_2, which can recover p_1 as $p_2 \oplus (p_1 \oplus p_2)$.

Another interesting example, but for wireless networks, is shown in Fig. 2.9(b) where nodes s_1 and s_2 are communicating with the help of a relay m. Each of these nodes is equipped with an omnidirectional antenna; s_1 and s_2 are too far away from each other for direct communication, but can both reach the relay m in the middle. Assume that s_1 and s_2 wish to exchange a pair of packets p_1 and p_2 (*e.g.*, a pair of public cryptographic keys). Without allowing network coding in the system, the exchange of the pair of packets (p_1 from s_1 and p_2 from s_2) requires four timeslots, since the relay node should receive two packets and retransmit them one at a time. Allowing network coding in the relay, the exchange is achieved within three rounds, without interference between concurrent transmissions.

From the above examples, it is clear that network coding offers gains in terms of throughput and delay, compared to simple routing. Furthermore, the application of network coding can also be beneficial for robustness to packet erasures, where network coding permits to exploit efficiently the network diversity.

2.6.1 Random Linear Network Coding

The most common way of implementing network coding is allowing the nodes to combine the packets with linear operations [34, 35]. In this case, a coded packet $\hat{p}$ is a linear combination of the packets stored in the buffer or cache, denoted as $\hat{\mathbf{p}} = \mathbf{a} \cdot \mathbf{P}$,

where **a** is a vector of coefficients drawn from a Galois field and **P** is a vector of the packets in the buffer or cache. Early network coding schemes [39, 41] require the use of complex algorithms to define the coefficients **a**. They assume that the sources have full knowledge of the network topology, which is an unrealistic assumption in large scale networks. Later works [20] have proposed random selection of the coding coefficients for generating the packets. If the coefficients are chosen in a large finite field of size q, random linear network coding can achieve the multicast capacity with a probability higher than

$$(1 - |\mathscr{C}|/q)^l,$$

where $|\mathscr{C}|$ is the number of clients. The parameter l corresponds to the maximum number of links receiving signals with independent randomized coefficients in any set of links constituting a flow solution from all sources to any client. If large finite fields are chosen, the probability of obtaining the multicast capacity tends to one. The equations system built by the network coded packets is with probability 99.6% decodable when the computations are performed in a finite field of size 2^{16} [20].

For further reading in network coding, the reader may refer to [9, 30, 48, 91]. In the next subsection, we present the state-of-the-art for the use of network coding in the NDN architecture, which is the main idea behind this thesis.

2.6.2 Network Coding in NDN

The application of network coding in ICN has been explored by Montpetit *et al.* [59] who proposed an architecture called *NC3N*. In this approach, Interests have a new field that contains the Data packet availability information of the client that sends the Interest, similar to the approach proposed by Sundararajan *et al.* [88] for TCP-based content retrieval. Nodes storing Data packets that match the name prefix of the received Interest, reply only if they can provide a network coded Data packet that conveys new information to the client. However, when there are multiple clients requesting the same content, *(i)* the aggregation of Interests is problematic, since Interests with the same name but coming from different clients contain different Data packets availability information; and *(ii)*, the pipelining of Interests, *i.e.*, sending multiple concurrent Interests for different Data packets, is also problematic, since all the pipelined Interests have the same Data packets availability information. The latter is undesirable as a node that has a matching Data packet will reply to multiple Interests with the same Data packet. These Data packets will be considered as duplicates by the

client, since only one of these will carry novel information with respect to the Data packets that are already available at the client.

Inspired by NC3N [59], Wu *et al.* [100] proposed CodingCache, where network coding is used to replace the Data packets in the cache of the network nodes. Due to the increased Data packet diversity in the network, the cache hit rate is improved. However, CodingCache suffers from the same drawbacks as NC3N, namely, the Interest aggregation and Interest pipelining are problematic. In the work presented by Llorca *et al.* [49], multicast delivery in network coding enabled ICN is optimized by finding the evolution of the Data packets that are cached in the network. However, this approach needs a central entity that is aware of the network topology and the Interests, which does not scale well with the number of network nodes because it requires signaling messages to continuously inform about changes in the topology.

The demonstrated benefits that network coding brings to ICN motivated researchers to study applications that can take advantage of network coding enabled NDN. Matsuzono *et al.* [55] have proposed L4C2, a network coding enabled mechanism for low latency, low loss video streaming over CCN. In L4C2, the network nodes estimate the acceptable delay and Data packet loss rate in their uplinks, adjusting the requested video quality accordingly. The clients first request non-network coded Data packets, and only request network coded Data packets when they detect Data packet losses. In this case, the benefits of network coding are only exploited when Data packet losses occur.

Bourtsoulatze *et al.* [14] presented a delivery scheme for scalable video transmissions over NDN. The approach uses Prioritized Random Linear Network Coding (PRLNC) [90] to account for different video layers that have unequal importance. To communicate the additional information needed to handle network coded Interests and Data packets, this approach proposes the use of Bloom filters [11]. Moreover, a rate allocation algorithm is formulated in order to choose the optimal rates of Interests sent by clients and routers, and an Interest forwarding strategy that implements the rate allocation algorithm is proposed. The evaluation shows that the approach achieves a close to optimal performance.

In Chapter 3 of this thesis we propose NetCodNDN, a network coding enabled NDN architecture, that enables optimal multipath content retrieval in multi-source and multi-client scenarios for data intensive applications. Moreover, the NetCodNDN architecture also solves the shortcomings of the aforementioned approaches. Specifi-

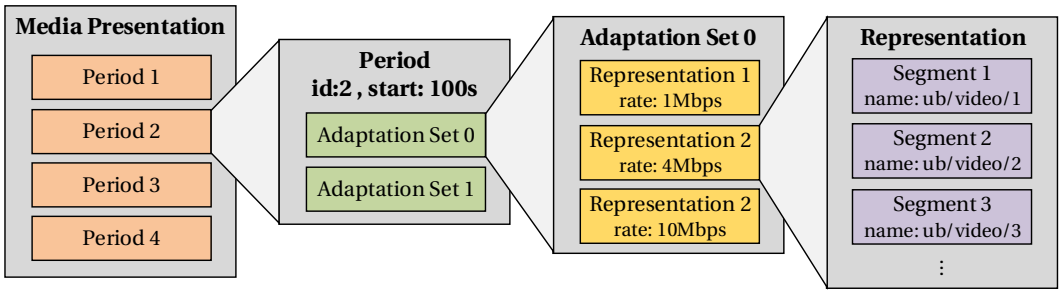

Figure 2.10: Representation of the Media Presentation Description (MPD) file.

cally, *(i)* it eliminates the need to include in the Interests the information about the Data packets available at the client, thus, simplifying Interest aggregation; *(ii)* it allows clients to efficiently send multiple Interests in parallel (*i.e.*, Interest pipelining), by modifying the way in which the nodes process the Interest messages; and *(iv)* it does not need centralized coordination or complicated inter-node signaling.

2.7 Video Streaming

Due to its predominance in the Internet traffic, in this thesis we choose video streaming as the data intensive application on which we base our study of the benefits that network coding brings to data intensive applications over NDN. In this section, we introduce the main concepts that enable dynamic adaptive video streaming over NDN. First, we describe the operation of Dynamic Adaptive Streaming over HTTP (DASH). Then, we show the similarities between NDN and DASH.

2.7.1 Dynamic Adaptive Streaming over HTTP

One of the most prominent video streaming techniques used nowadays is adaptive video streaming, and, in particular, DASH [37, 85]. One of the main characteristics of DASH is that the clients are in control of the streaming logic, deciding the bitrate and resolution of the streamed video. To enable the video quality adaptation by the clients, each video v is encoded with different parameters (*e.g.*, bitrate, resolution, *etc.*), creating a set of *representations* $\mathcal{Q}$. The different representations can be generated by using any video encoder (*e.g.*, AV1 [65], HEVC/H.265 [86], *etc.*), as it is not fixed in the DASH standard. The video data in each representation $q \in \mathcal{Q}$ is divided into a set of *segments* $\mathcal{Z}$. Every segment $z \in \mathcal{Z}$ has the same duration. This allows

clients to request segments belonging to the representation that better adapts to their current network conditions, display capabilities, *etc.* Hence, the clients can switch to a different representation after receiving each segment if the network conditions change, for example. To inform the clients about the offered video representations, a file called the Media Presentation Description (MPD) is associated with each video v. As it can be seen in Fig. 2.10, the MPD file contains information about the *periods*, which are fractions of the video that may have a different set of representations, the *adaptation sets*, which may have different sets of representations for different devices, the available representations for each adaptation set, and the segments that compose each representation. In this thesis we consider that there exists a single period and a single adaptation set, thus we are only interested in the sets of representations $\mathcal{Q}$ and segments $\mathcal{Z}$ available in the MPD file. A client that is interested in receiving the video v should first request the MPD file associated with this video. Then, after receiving and parsing the MPD file of the video v, the client knows what representations are available and what names it should use to request the video segments. Each particular video segment is identified with a name $n \leftarrow \texttt{v/q/z}$ that is composed of the ID v of the video to which the segment belongs, the representation q in which the segment has been encoded and the segment ID z.

2.7.2 Dynamic Adaptive Streaming over NDN

The similarities in the content retrieval mechanisms of DASH and NDN have attracted the attention of the research community [25, 45, 46, 73, 95, 98, 99]. Detti *et al.* [25] proposed a cooperative adaptive video streaming application for CCN. In this application, mobile users download video segments from the sources over the cellular network and also from other mobile users that are connected through the Wi-Fi network. The results show that by exploiting users' cooperation, the users can drastically reduce the amount of data downloaded over the cellular network, which may result in cost reductions. In a system integrating CCN and DASH [46], the *version* component of the CCN content naming is used to name the different representations of a DASH segment, and the *segment* component of the CCN content naming is used to divide segments into Data packets that fit into lower level Maximum Transmission Units (MTU). Clients request video segments following the same procedures as in the CCN architecture. However, this DASH-based video streaming proposal [46] uses NDN as communication protocol, it suffers from the drawbacks for data intensive applications in NDN presented in Section 1.1.

As presented in Section 1.2, in this thesis we propose to use network coding in order to alleviate the aforementioned issues. On top of that, in this thesis, we propose a novel DASH-based video streaming architecture for network coding enabled that enables optimal video streaming in multi-source and multi-client scenarios. It is worth noting that the Internet Research Task Force (IRTF) addresses the adaptation of current video streaming mechanisms to the ICN architecture in the RFC7933 [99]. It also defines some use cases for video streaming and their requirements, identifying the main issues associated with these streaming mechanisms in ICN. Moreover, RFC7933 [99] also recognizes the potential benefits that network coding could bring to video streaming over NDN, considers its study as an important next step for the ICN community.

2.8 Caching in Named Data Networking

As described in Section 2.4, in NDN the intermediate routers are able to cache the received Data packets in order to reply to future Interests for the same content. This pervasive caching concept proposed by NDN has enabled the study and proposal of multiple approaches to improve the caching efficiency. However, having caches in all the network nodes is not always necessary to yield the full benefits that caching brings to the data retrieval process. In particular, it is agued that enabling caches only at the edge of the network may achieve performance improvements similar to having caches in every node [24, 26, 87].

The caching process can be divided in two phases: first, a *placement* algorithm that decides which Data packets should be cached, and the *replacement* algorithm that decides which data packets should be evicted from the cache when it is full and a new Data packet should be inserted. Some of the basic placement algorithms are Leave Copy Everywhere (LCE), where all the Data packets received by a router are placed into the cache, and Leave Copy Down (LCD) [44], where each time a router registers a cache-hit, it flags the Data packet such that it is cached one hop closer to the user. A very popular approach is ProbCache [74], where the routers cache Data packets according to a certain probability that depends on the caching resources available along the path. In this way, the caching resources are better utilized, reducing the number of duplicate Data packets cached in the network.

More sophisticated placement algorithms consider content popularity in order to decide which Data packets a router should place in its cache [3, 10, 19, 47, 97, 98, 103].

MPC [10] is a placement algorithm that proposes the use of a popularity threshold at each router. As soon as the popularity of a Data packet reaches the threshold, the router sends a suggestion to its neighbors to cache the popular Data packet. The main drawback of this approach is the increased traffic in the network due to the suggestion messages, which can be potentially high, since the granularity is at the Data packet level.

VIP [103] is a framework for joint Interest forwarding and Data packet caching. This scheme uses a virtual control plane that operates on the Interest rate, and an actual plane which handles Interests and Data packets. It is shown that the design of joint algorithms for routing and caching is important for NDN, and, thus, this scheme proposes distributed control algorithms that operate in the virtual control plane with the aim of increasing the number of Interests satisfied by in-network caches.

PopCaching [47] is a popularity-based caching policy in which the popularity is computed online, without the need for a training phase. This makes PopCaching robust to dynamic popularity settings. However, PopCaching is designed for traditional caching systems with a single cache in the path, while in this paper we are interested in networks of caches.

WAVE [19] is a placement algorithm that determines the number of Data packets that should be cached for a given file with the help of an access counter. The number of Data packets to cache increases exponentially with the value of the access counter. The main idea of WAVE, which partially caches a content object according to the local popularity, is also shared by the caching policy that we propose in this paper. However, this approach does not facilitate edge caching, since the most popular data is cached closer to the source and slowly moves towards the edges as the number of requests increases.

Progressive [3] is another partial caching algorithm, which exploits the content popularity to decide how many Data packets should be cached for each name prefix. The cache placement decision is taken when the Interests are received, which helps to cache the most popular content at the network edge. However, this approach lacks a replacement algorithm, and hence it cannot be deployed when the cache capacity is limited.

None of the approaches above consider the use of network coding [7], and they are evaluated in single-path scenarios. Given the benefits that network coding brings

to multipath communications in NDN [59, 75, 78, 80], some approaches have been proposed to improve the benefits of caching in network coding enabled NDN architectures [49, 97, 100]. NCCAM [49] and NCCM [97] propose optimal solutions to the problem of efficiently caching in network coding enabled NDN. However, both approaches need a central entity that is aware of the network topology and the Interests, which does not scale well with the number of network nodes. CodingCache [100] is an eviction policy in which before evicting a Data packet, the routers apply network coding to the Data packet by means of combining it with other Data packets with the same name prefix that will remain in the cache. Due to the increased Data packet diversity in the network, the cache-hit rate is improved. However, in CodingCache the clients should include into the Interests information regarding the Data packets that they have received. This *(i)* increases the size of the Interests, since the Data packet availability information may be of considerable size, *(ii)* complicates the Interest aggregation procedure, since the information about the available Data packets for each client needs to be aggregated with the Interests, and *(iii)* complicates the Interest pipelining procedure, since a client that sends multiple Interests in parallel with the same Data packet availability information may receive duplicate Data packets.

In this Thesis we propose PopNetCod, a popularity based caching policy for network coding enabled NDN. PopNetCod is a distributed caching policy, in which each router measures the popularity of the Interests for content objects that it receives, and uses this information to decide which Data packets it will cache or evict from the content store. This naturally enables partial caching of content objects. The placement decision takes place when processing the Interests. In case that the router decides to cache the Data packet that will come as reply to this Interest, it signals the other routers in the path with a flag on the Interest. Taking the placement decision at the Interest processing stage helps to keep the most popular Data packets closer to the network edges, which is beneficial [24, 26, 87].The replacement decision is taken when processing a Data packet that should be cached and the content store is full, also using popularity information.

3

A Network Coding Enabled NDN Architecture

3.1 Introduction

It is clear from Chapter 1 that the integration of network coding into the Named Data Networking (NDN) architecture presents benefits for data intensive applications. Specifically, it enhances multipath data retrieval in multi-source and multi-client scenarios, alleviates throughput degradation at the clients when bottlenecks are present in the network, and improves the resilience to packet loss.

In order to study in detail these benefits and quantify them, an architecture that integrates network coding into the NDN architecture is needed. In particular, the needed architecture should enable network coding in NDN in a way that it can handle the large number of Data packets that data intensive applications require. Moreover, it should support multi-client and multi-source scenarios in diverse network topologies, since these scenarios are becoming more important.

Therefore, this Chapter introduces NetCodNDN [78], a network coding enabled NDN architecture, based on the CCN/NDN architecture [38, 104]. The proposed architecture enables optimal multipath content retrieval in multi-source and multi-client scenarios for data intensive applications. Moreover, the NetCodNDN architecture also solves the shortcomings of the approaches presented in Chapter 2. We have implemented NetCodNDN by making the necessary changes to the CCNx [17] codebase, and performed experiments to compare it to unmodified CCNx. The results demonstrate that NetCodNDN offers large gains in terms of the time needed to retrieve the original content object. Moreover, it improved robustness to packet losses and permits to exploit more efficiently the available network resources in multi-source and multi-client scenarios. To the best of our knowledge, this is the first practical implementation that enables network coding in the CCNx codebase.

This Chapter is organized as follows. We start by describing the integration of network coding into NDN and the challenges it poses. We then describe the generation of network coded Data packets, the basic transmission unit in our proposed architecture. Then, we present the core of the NetCodNDN architecture: the data structures and algorithms used by the routers to process network coded Interests and Data packets. Finally, we present the evaluation of the NetCodNDN architecture.

3.2 Network Coding Enabled NDN

As discussed in Chapter 1, the shortcomings of the NDN architecture in multi-client and multi-source scenarios for data intensive applications can be resolved by enabling network coding [7]. With network coding, the Data packets delivered to the clients are coded by means of combining the Data packets available at sources and routers prior to being forwarded. Hence, when network coding is enabled in NDN, the network coded Data packets contain information from all the Data packets that have been combined to generate them. The key idea behind introducing network coding in NDN is that clients no longer need to request specific Data packets, but rather network coded Data packets, as they all have equivalent information. Therefore, the nodes do not need to coordinate the faces where they forward the Interests, which enhances the network bandwidth utilization.

Differently from NDN, where an Interest $i_{n,j}$ requests a specific Data packet $p_{n,j}$ with name prefix n and Data packet ID j, in a network coding enabled NDN, an Interest

$\hat{i}_n$ requests a network coded Data packet $\hat{p}_n$, without specifying the particular Data packet ID j. In this case, the set of Interests needed to retrieve $\hat{\mathscr{P}}_n$ is $\hat{\mathscr{I}}_n = \{\hat{i}_n\}$, *i.e.*, the set contains a single Interest. To retrieve the demanded content, each client sends the Interest $\hat{i}_n$ at least $|\mathscr{P}_n|$ times. Note that more than $|\mathscr{P}_n|$ Data packets may be needed, as network coded Data packets are generated by randomly combining the Data packets with name prefix n. Hence, with some small probability when coding is done in a large finite field, these Data packets can be linearly dependent. Furthermore, due to packet losses, additional Interests may need to be sent to compensate for the lost packets. Any node v can reply to these Interests with network coded Data packets. The network coded Data packets are generated by combining the set of Data packets $\mathscr{P}_n^v$ that are available in the CS or the repository of the node v and that match the name prefix n.

The Data packet $\hat{p}_n$ can be considered as a vector $\hat{\mathbf{p}}_n$, where each element of the vector belongs to a finite field. Then, the set of network coded Data packets $\mathscr{P}_n$ can be expressed as a matrix $\hat{\mathbf{P}}_n$ where each row corresponds to a Data packet $\hat{\mathbf{p}}_n$. The operations performed by the node v to generate a new network coded Data packet $\hat{\mathbf{p}}_n$ can be expressed as

$$\hat{\mathbf{p}}_n = \mathbf{A} \cdot \mathbf{P}_n^v, \tag{3.1}$$

where $\mathbf{A}$ is a matrix of coding coefficients drawn uniformly at random from a finite field, and $\mathbf{P}_n^v$ is the matrix formed with the set of Data packets $\mathscr{P}_n^v$. When the coding coefficients in $\mathbf{A}$ are randomly chosen from a large enough finite field, the generated Data packets have a high probability of being linearly independent with respect to the Data packets previously generated, and, thus, being innovative [35]. To decode the original Data packets that compose $\mathscr{P}_n$, a client should collect $|\mathscr{P}_n|$ innovative network coded Data packets $\hat{p}_n$.

3.3 Challenges of Enabling Network Coding in NDN

As discussed in Chapter 1, enabling network coding in NDN brings benefits that can potentially improve the performance of content object retrieval under multi-source and multi-client scenarios. However, some issues arise when the Interests do not specify the Data packet ID.

One of the issues that arises is that any node that has a single network coded Data packet $\hat{p}_n$ cached in its CS will reply with this Data packet to any Interest $\hat{i}_n$, as the name prefix in the Interest matches that of the cached Data packet $\hat{p}_n$. This is undesirable, since the routers will always reply with the same cached Data packet $\hat{p}_n$, while clients need to receive $|\mathscr{P}_n|$ innovative network coded Data packets in order to decode the original content object. Therefore, the routers need a way to determine when they cannot provide a network coded Data packet that is innovative to the client, and thus a new network coded Data packet has to be retrieved. In the previous work from Montpetit *et al.* [59], the authors propose to solve this problem by allowing the clients to include information about the coded Data packets they have collected so far. Routers reply to an Interest only if they can provide innovative information. However, it is not clear how routers can aggregate Interests with different information from the clients.

Another challenge that emerges when network coding is enabled in NDN is related to the *pipelining* procedure, *i.e.*, a client sending multiple concurrent Interests for different Data packets of the same content object. In the original NDN when a node receives an Interest that it cannot satisfy with content stored in its CS, the node checks its PIT. If the node finds an entry in the PIT indicating that an Interest for the same name has been received previously over the same face, it considers this new Interest as a duplicate and do not forward it. Since Interests for different Data packets have different names, as the Data packet ID is appended to the name prefix, pipelining is supported. However, when network coding is enabled in NDN, concurrent Interests for different network coded Data packets of the same content object have the same name. Therefore, pipelining can not be supported trivially, as all Interests with the same name will be considered duplicated.

3.4 The NetCodNDN Architecture

In this Section, we describe our proposed NetCodNDN architecture, a practical implementation of a network coding enabled NDN architecture, based on the CCN [38] and NDN [104] architectures. We start by defining the data segmentation and naming scheme in the proposed architecture. Then, we describe the proposed changes to the data structures of an NDN node (*i.e.*, the Content Store (CS) and the Pending Interest Table (PIT)) that render them able to deal with network coded Data packets and Interests. Finally, we describe how Interests and Data messages are processed in

NetCodNDN.

3.4.1 Content Object Fragmentation

As in the NDN architecture, in NetCodNDN the content objects are split into smaller Data packets, $P_n = \{p_{n,1}, \ldots, p_{n,M}\}$, that fit into a Maximum Transmission Unit (MTU). As presented in Section 3.2, the Data packet $p_{n,j}$ can be represented as a vector $\mathbf{p_{n,j}}$. To facilitate the deployment of network coding in practical settings [20], each Data packet $\mathbf{p_{n,j}}$ is prepended with an encoding vector to inform the routers and clients about the coding operations that the network coded Data packet has been subjected to. At the sources, the initial value of this vector is set to be the jth unit vector, which has value 1 in the jth position and 0 otherwise.

Prepending encoding vectors to the Data packets introduces a communication overhead that consumes network resources, especially when the number of Data packets $|P_n|$ is large. To limit this overhead, we adopt the concept of *generations* [20], where the original set of Data packets that compose $\mathscr{P}_n$ is divided into smaller groups of Data packets, which are known as generations. The coding operations are restricted only among Data packets that belong to the same generation, in order to reduce the network coding overhead and make it appropriate for time-constrained applications. The set of Data packets that form the generation g is denoted as $\mathscr{P}_{n,g}$, where g is the generation ID. Thus, $\mathscr{P}_n = \cup_{g=1}^{G} \mathscr{P}_{n,g}$, where G is the total number of generations. To avoid mixing Data packets from different generations, the generation ID, g, is added to the name of the Data packet. The size of the generation controls the tradeoff between the overhead required to communicate the encoding vector and the Data packet diversity. Note that the size of the encoding vectors prepended to each Data packet is equal to the size of the generation g, *i.e.*, $|\mathscr{P}_{n,g}|$, as the network coded Data packets may potentially carry information from all the Data packets in $\mathscr{P}_{n,g}$. Overall, the encoding vectors do not pose any limitations to our system as there are approaches to compress them efficiently [53, 89].

The node v generates network coded Data packets by randomly combining the set of Data packets $\mathbf{P}_{n,g}^{v}$ with name prefix n, g that are stored in their CS. Thus,

$$\hat{\mathbf{p}}_{n,g} = \mathbf{a} \cdot \mathbf{P}_{n,g}^{v} = \sum_{j=1}^{|\hat{\mathbf{P}}_{n,g}^{v}|} a_j \cdot \hat{\mathbf{p}}_{n,g}^{(j)}, \tag{3.2}$$

where $\mathbf{a} = \{a_1, \ldots, a_L\}$ is a vector of coding coefficients randomly selected from a finite field. Note that the sum and multiplication operations in (3.2) are finite field operations.

3.4.2 Data Packet Naming

From the discussion above, it is obvious that the naming in NetCodNDN should have two additional components compared to the NDN architecture, namely the encoding vector and the generation ID g. For example, let us consider a content object P_n with name n = `/provider/videos/largevideo.h264`, that is partitioned into G generations of four Data packets each. Thus, in NetCodNDN the first Data packet of the gth generation, associated with the coding vector $[1,0,0,0]$, is named `/provider/videos/largevideo.h264/g/1000`. Note that the proposed naming scheme is compatible with the original NDN and can support the delivery of non-network coded Data packets.

3.4.3 Data structures in NetCodNDN

Each node in the NetCodNDN architecture has a network layer forwarder, called NetCodNDN forwarder, that is in charge of *(i)* routing Interests towards the sources, *(ii)* forwarding Data packets back to the clients, and *(iii)* applying network coding operations on the Data packets before forwarding them.

In the following, we describe the data structures maintained by the NetCodNDN forwarder: the Forwarding Information Base (FIB), the Content Store (CS), and the Pending Interest Table (PIT).

Forwarding Information Base

NetCodNDN uses the FIB model provided by NDN [104], and described in Chapter 2.

Content Store

NetCodNDN uses the CS model provided by NDN [104], and described in Chapter 2, for both, network coded and traditional Data packets. However, in order to support the creation of network coded Data packets, the methods to insert new Data packets

into the CS and to get Data packets that it holds are modified.

As described in Section 3.4.1, when a node v with a NetCodNDN forwarder receives an Interest $\hat{i}_{n,g}$, it can use the set of Data packets $\hat{\mathscr{P}}^{v}_{n,g}$ that are stored in its CS to generate a network coded Data packet and reply to the Interest. Since both, traditional Data packets $p_{n,j}$ and network coded Data packets $\hat{p}_{n,g}$ are stored as single entries in the CS, the NetCodNDN CS `GetCS(`n,g`)` method returns a set of Data packets that match the name prefix (n,g), rather than a single Data packet as in the NDN CS described in Chapter 2. The new methods provided by the NetCodNDN CS are described below.

- `GetCS(`n,g`)` – Gets the set of Data packet $\hat{\mathbf{P}}^{v}_{n,g}$ that match the name prefix (n,g) that are cached in the CS of the node v. This function iterates over all the CS until to find all the Data packets that match the name prefix (n,g).
- `InsertCS(`$p_{n,g}$`)` – Inserts a new Data packet $p_{n,g}$ into the CS of the node v. Note that different network coded Data packets with name prefix (n,g) have a different encoding vector appended to the name, as described in Section 3.4.2, which makes them unique in the CS.

Pending Interest Table

Recall from Chapter 2 that each PIT entry $t_{n,j}$ has two components for each face f, *(i)* an *in-record* $t^{f(in)}_{n,j}$ that keeps track of the Interests that arrived over face f and that have not been satisfied, and *(ii)* an *out-record* $t^{f(out)}_{n,j}$ that keeps track of the Interests that have been forwarded over face f, and that are still pending. Each in-record $t^{f(in)}_{n,j}$ takes the value "1" if an Interest with name prefix (n,j) has been received over face f and it has not been satisfied, or the value "0" otherwise. If an Interest with name prefix (n,j) arrives to a node in which the value of $t^{f(in)}_{n,j}$ is "1", the Interest is considered duplicated and not further processed.

As described in Section 3.3, Interest pipelining is not trivially supported in NetCodNDN. This is because, since in the NetCodNDN architecture an Interest $\hat{i}_{n,g}$ request any coded Data packet $\hat{p}_{n,g}$ from the set of Data packets with name prefix (n,g), where g is the generation ID, rather than particular Data packets $p_{n,j}$, the first received Interest will set the value of $t^{f(in)}_{n,g}$ to "1", and all the subsequent pipelined Interests $\hat{i}_{n,g}$ will be considered as duplicated.

To enable Interest pipelining in the NetCodNDN architecture, the design of the PIT has

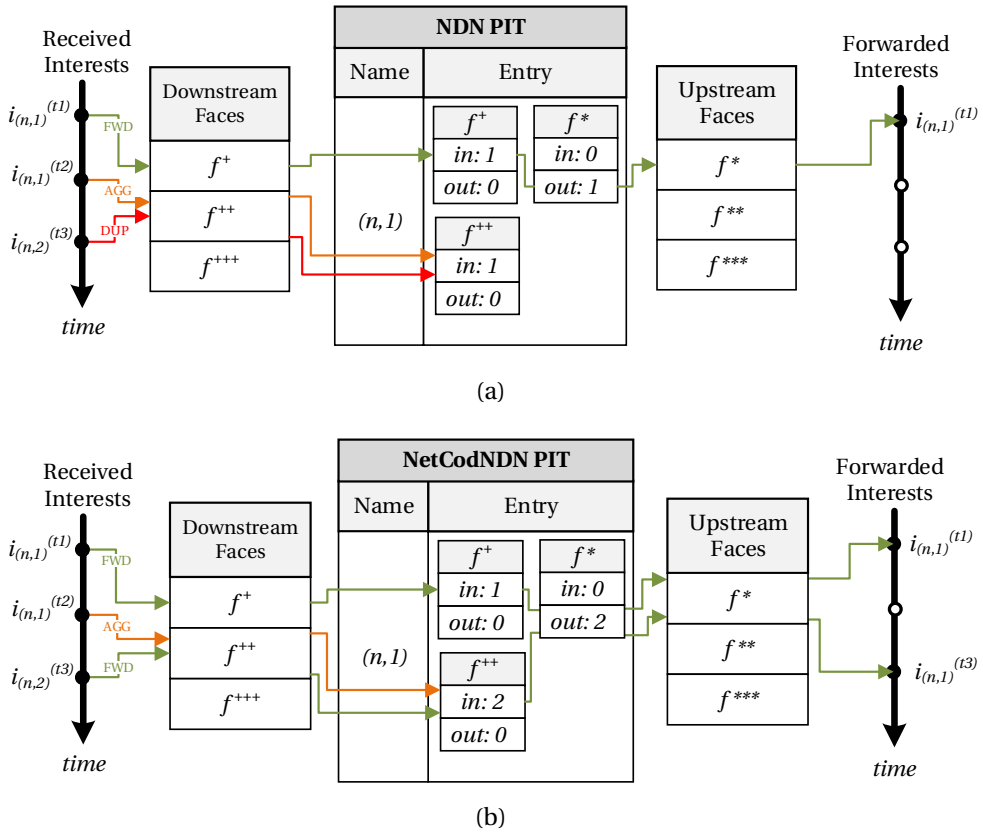

Figure 3.1: Comparison of the PIT in (a) NDN and in (b) NetCodNDN.

to be modified with respect to that of the original NDN forwarder. In the redesigned NetCodNDN PIT, the PIT in-record $t_{n,g}^{f(in)}$ and the out-record $t_{n,g}^{f(out)}$ are counters which can take values higher than "1", *i.e.*, there could be multiple pending Interests with name prefix (n,g) for the face f. In this way, NetCodNDN nodes consider all pipelined Interests as requests for more network coded Data packets, rather than duplicated Interests.

To illustrate how the NetCodNDN PIT differs from the NDN PIT, let us consider the example presented in Fig. 3.1. In both NDN and NetCodNDN, the first two Interests are processed in a similar way: the Interest $i_{n,g}^{(t1)}$ is forwarded and the Interest $i_{n,g}^{(t2)}$ is aggregated. However, the third Interest $i_{n,g}^{(t3)}$ is processed differently in the two schemes: in NDN, it is considered duplicated since an Interest for the same name has been received over face f^{++} previously; in NetCodNDN, the Interest is forwarded since it is considered as a request for an additional Data packet with name prefix (n,g) from the face f^{++}.

The NetCodNDN PIT provides methods to insert information about new Interests, and to get the PIT entry associated with a particular name prefix. In particular, the following two methods are provided:

- `GetPIT((n,g))` – Gets the PIT entry $t_{n,g}$ associated with the name prefix (n,g), if it exists.
- `InsertInPIT((n,g),f)` – Increases by 1 the value of $t_{n,g}^{f(in)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, it is created by this method.
- `RemoveInPIT((n,g),f)` – Decreases by 1 the value of $t_{n,g}^{f(in)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, this method does nothing.
- `InsertOutPIT((n,g),f)` – Increases by 1 the value of $t_{n,g}^{f(out)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, it is created by this method.
- `RemoveOutPIT((n,g),f)` – Decreases by 1 the value of $t_{n,g}^{f(out)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, this method does nothing.

3.4.4 Interest Processing

Similarly to NDN, in the NetCodNDN architecture the data communication is triggered by the clients who send Interest messages $\hat{i}_{n,g}$ for data with name prefix (n,g). In the proposed NetCodNDN architecture, the Interests have a *NetworkCodingAllowed* field that takes the value "1" when network coded packets are expected, otherwise, the field is not present or its value is set to "0". If the `NetworkCodingAllowed` field is set to the value "1", the NetCodNDN Interest processing procedures are invoked. If the `NetworkCodingAllowed` field is not present or set to the value "0", the Interests are treated following the original NDN procedures [4]. When a node v receives an Interest $\hat{i}_{n,g}$ with the *NetworkCodingAllowed* field activated over face f, it either *(i)* replies to the Interest with a network coded Data packet generated by combining the Data packets in its CS; *(ii)* forwards the Interest to its neighboring nodes, to receive an innovative network coded Data packet; or *(iii)* waits for a new network coded Data packet to arrive, if a previously received Interest with the same name prefix has

Algorithm 1 Interest Processing in NetCodNDN

Require: $\hat{i}_{n,g}, f, \mathbf{P}^{v}_{n,g} \leftarrow$ `GetCS`(n,g)

1: **if** `rank`$(\hat{\mathbf{P}}^{v}_{n,g}) = |\hat{\mathscr{P}}_{n,g}|$ **then** (*Generation is decodable*)
2: $\quad \xi^{f}_{n,g} = |\hat{\mathscr{P}}_{n,g}|$
3: **else**
4: $\quad \sigma^{f}_{n,g}$ = number of Data packets with name prefix (n,g) sent over face f.
5: $\quad \xi^{f}_{n,g} =$ `rank`$(\hat{\mathbf{P}}^{v}_{n,g}) - \sigma^{f}_{n,g}$
6: **end if**
7: **if** $\xi^{f}_{n,g} > 0$ **then**
8: $\quad \hat{\mathbf{p}}_{n,g} \leftarrow \sum_{j=1}^{|\hat{\mathbf{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
9: $\quad$ Send Data packet $\hat{p}_{n,g}$ over face f
10: **else**
11: $\quad$ `InsertInPIT` $((n,g), f)$
12: $\quad t_{n,g} \leftarrow$ `GetPIT` $((n,g))$
13: $\quad \epsilon_{n,g} = \sum_{f'}^{\mathscr{F}^{v}_{n,g}} t^{f'(out)}_{n,g}$ (*Total number of Interests $\hat{i}_{n,g}$ forwarded*)
14: $\quad$ **if** $\epsilon_{n,g} \leq t^{f(in)}_{n,g}$ **then**
15: $\quad\quad$ Forward the Interest $\hat{i}_{n,g}$ over face f_{out}.
16: $\quad\quad$ `InsertOutPIT` $((n,g), f_{out})$
17: $\quad$ **else**
18: $\quad\quad$ Wait for new Data packets.
19: $\quad$ **end if**
20: **end if**

already been forwarded. This procedure is further explained below and summarized in Algorithm 1.

Replying to an Interest — The node v replies to an Interest $\hat{i}_{n,g}$ when *(i)* it has collected $|\hat{\mathscr{P}}_{n,g}|$ innovative network coded Data packets, meaning that the generation g is decodable (line 1); or when *(ii)* a network coded Data packet generated by the node v has high probability to be innovative for the neighbor node connected through face f over which the Interest arrived. The number of network coded Data packets that can be generated by the node v and that have a high probability to be innovative is given by $\xi^{f}_{n,g} =$ `rank`$(\hat{\mathbf{P}}^{v}_{n,g}) - \sigma^{f}_{n,g}$ (line 5). The parameter $\sigma^{f}_{n,g}$ denotes the number of network

coded Data packets that have been sent over face f. When $\xi^{f}_{n,g}$ is greater than 0 (line 7), the node v generates a new network coded Data packet $\hat{p}_{n,g} = \sum_{j=1}^{|\hat{\mathscr{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$ (line 8) and sends it over face f (line 9).

Forwarding an Interest — If the number of network coded Data packets that can be generated by the node v and that have a high probability to be innovative, $\xi^{f}_{n,g}$, is equal to 0 (line 10), the node v needs to receive a new innovative Data packet that increases the rank of $\hat{\mathbf{P}}^{v}_{n,g}$ before it is able to reply to the received Interest $\hat{i}_{n,g}$. Before forwarding an Interest, the node v checks its PIT. In order to support Interest pipelining, *i.e.*, sending multiple concurrent Interests for different Data packets of the same content object, the PIT lookup procedure of the NetCodNDN forwarder is different from that of the NDN forwarder. First, the node v increases by 1 the value of $t^{f(in)}_{n,g}$ in the PIT entry $t_{n,g}$ (line 11). Then, node v computes the number $\epsilon_{n,g}$ of innovative network coded Data packets with name prefix (n,g) that it is expecting to receive before the Interest $\hat{i}_{n,g}$ expires. To compute $\epsilon_{n,g}$, node v needs to take into consideration the Interest and Data packet loss rate and delays, among other variables. For the sake of simplicity, the NetCodNDN forwarder assumes that any forwarded Interest brings an innovative Data packet before its expiration. This assumption is similar to the one made by the original NDN forwarder, where received Interests are not further forwarded if a PIT entry matching the name of the Interest is found, since the previously forwarded Interest is expected to bring the requested Data packet. This is because the NDN forwarder also considers that every forwarded Interest will bring the requested Data packet before its expiration. In this case, $\epsilon_{n,g}$ is equal to the total number of Interests with name prefix (n,g) that have been forwarded by the node v over all its faces, *i.e.*, the sum of all the out-records in the PIT entry $t_{n,g}$ (line 13). Finally, the node forwards the Interest $\hat{i}_{n,g}$ if the number of innovative network coded Data packets with name prefix (n,g) that it is expecting to receive is less than or equal to the number of Data packets with name prefix (n,g) that are pending to be sent over face f, *i.e.*, $\epsilon_{n,g} \leq t^{f(in)}_{n,g}$ (lines 14 - 16).

Waiting for a new network coded Data packet — If $\epsilon_{n,g} > t^{f(in)}_{n,g}$, the node v does not forward the Interest $\hat{i}_{n,g}$ and waits for a new network coded Data packet to arrive, as it expects to receive enough network coded Data packets to satisfy all the pending Interests, including the received Interest (lines 17 - 18).

Algorithm 2 Data packet processing in the NetCodNDN forwarder

Require: $\hat{p}_{n,g}, f$

1: $t_{n,g} \leftarrow$ `GetPIT ((`n,g`))`
2: **if** $t_{n,g}^{f(out)} = 0$ **then** (*Unsolicited*)
3: Discard $\hat{p}_{n,g}$
4: **else**
5: `RemoveOutPIT((`n,g`),`f`)`
6: **if** `rank(`$\hat{\mathbf{P}}_{n,g}^{v} \cup \hat{p}_{n,g}$`) >` `rank(`$\hat{\mathbf{P}}_{n,g}^{v}$`)` **then**
7: `InsertCS (`$\hat{p}_{n,g}$`)`
8: **for all** $f' \in t_{n,g}$ **do**
9: **if** $t_{n,g}^{f'(in)} > 0$ **then**
10: $\hat{p}_{n,g}^{*} = \sum_{j=1}^{|\hat{\mathscr{P}}_{n,g}^{v}|} a_j \cdot \hat{\mathbf{p}}_{n,g}^{(j)}$
11: Send the Data packet $\hat{p}_{n,g}^{*}$ over face f'
12: $\sigma_{n,g}^{f} \leftarrow \sigma_{n,g}^{f} + 1$
13: `RemoveInPIT((`n,g`),`f`)`
14: **end if**
15: **end for**
16: **else**
17: Discard $\hat{p}_{n,g}$
18: **end if**
19: **end if**

3.4.5 Data Message Processing

When a node receives a network coded Data packet, it follows the procedure outlined in Algorithm 2 and described below.

When a node v receives a network coded Data packet $\hat{p}_{n,g}$ over face f, it first determines whether this Data packet was expected or if it was unsolicited. The node v accomplishes this by looking at its PIT (line 1). If the number of Interests sent over the face f is 0, *i.e.*, $t_{n,g}^{f(out)} = 0$, the node considers the Data packet as unsolicited and does not transmit it further (lines 2 - 3). Otherwise, if the Data packet was expected (line 4), the node v decreases the value of $t_{n,g}^{f(out)}$ by 1 (line 5) and determines if the

Data packet $\hat{p}_{n,g}$ is innovative. The Data packet $\hat{p}_{n,g}$ is innovative for the node v if it is linearly independent with respect to all the Data packets in the CS of the node v, $\hat{\mathscr{P}}^{v}_{n,g}$, *i.e.*, if it increases the rank of $\hat{\mathbf{P}}^{v}_{n,g}$ (line 6). If the Data packet $\hat{p}_{n,g}$ is innovative, the node v inserts it into its CS (line 7). Then, for each face f' indicated in the PIT entry $t_{n,g}$ (line 7), the node v determines if any pending Interest should be satisfied, *i.e.*, $t^{f'(in)}_{n,g} > 0$. If an Interest should be satisfied for the dace f', the node generates a new network coded Data packet $\hat{p}^{*}_{n,g} = \sum_{j=1}^{|\hat{\mathscr{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$ (line 10) and sends it over that face (line 11). After this, the node updates the number of Data packets that have been sent over face f' (line 12) and decreases by 1 the number of pending Interests for the face f' (line 13). If the Data packet is non-innovative, it is discarded by node v (lines 16 - 17).

3.4.6 Complexity

It is important to note that network coding adds complexity to both the Interest and Data message processing in NetCodNDN. Performing algebraic operations on the Data packets before forwarding them adds, effectively, some complexity to the node. In particular, as we have seen in Section 3.4.1, a node generates a new coded packet as $\hat{p}_{n,g} = \sum_{j=1}^{|\hat{\mathscr{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$. If we consider that the symbol size is 2^8 and that Data packets are of size $|\hat{\mathbf{p}}|$ symbols, each time a node needs to generate a new coded packet, it performs $|\hat{\mathbf{p}}| \cdot |\hat{\mathscr{P}}^{v}_{n,g}|$ multiplications and $|\hat{\mathbf{p}}| \cdot (|\hat{\mathscr{P}}^{v}_{n,g}| - 1)$ additions. This complexity does not pose limitations to our scheme as there are efficient implementations of network coding [70]. Moreover, it has been shown [106] that a network coding coder and decoder can operate at wire-speed with rates of up to 1000Mbps.

3.5 Evaluation

In this Section, we evaluate the performance of NetCodNDN in various scenarios, and compare the results to the performance of the standard NDN. It is worth noting that, while multiple approaches proposing the integration of network coding into NDN exist in the literature [49, 59, 97, 100], none of them provides neither a implementation that could be used in our experiments, nor a detailed description of the architecture that enable us to implement it. In the remaining of this Section, we first describe the simulation setup. Then, we evaluate the performance of NetCodNDN in the butterfly network. This toy network provides a controllable environment which permits to verify

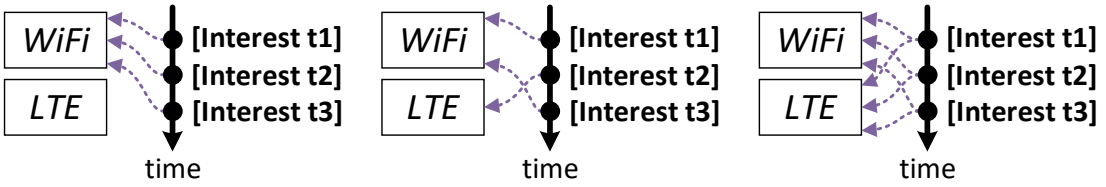

Figure 3.2: Interest distribution across the diffreren faces with the (a) *default*, (b) *loadsharing*, and (c) *parallel* Interest forwarding strategies.

the expected behavior of NetCodNDN, and facilitates the illustration of its benefits. Finally, we present the simulation results in a more realistic network topology, which is generated based on real network measurements taken from the PlanetLab project [72].

3.5.1 Evaluation Setup

We implemented NetCodNDN by integrating the changes to the NDN architecture described in Section 3.4 into the CCNx 0.8.2 [17] codebase, and we compare its performance to that of the unmodified CCNx. The network topology is simulated using the NS-3 network simulator [63]. The software forwarders for CCNx and NetCodNDN are installed on NS-3 nodes using the Direct Code Execution framework (DCE) [64].

We consider that the clients are interested in a content object composed of 100 packets, *i.e.*, $|\hat{\mathscr{P}}_n| = 100$. The size of each Data packet is 5KB. The Data packets are stored in a set of sources that are connected to the clients through a network of routers. We consider that the routers have sufficient CS capacity to cache all the incoming Data packets. We assume that the 100 source Data packets comprise a single generation, *i.e.*, $|\hat{\mathscr{P}}_{n,g}| = |\hat{\mathscr{P}}_n|$ and $G = 1$. The finite field in which the network coding operations are performed is of size 2^8. In order to evaluate our architecture in a challenging scenario, we consider that all the clients send Interests during the same interval of time. In this way, we demonstrate that by using our architecture, the nodes are able to aggregate Interests adequately.

For the evaluation of CCNx, we consider the three main Interest forwarding strategies implemented in CCNx 0.8.2, and described in [16]:

- The *default* (DS) strategy *"prefers the fastest responding face, and performs experiments to determine if other faces can provide faster response. This strategy*

also operates efficiently in environments where link quality changes or a face becomes unresponsive, but does not make use of multiple paths or sources" [16]. In the example shown in Fig. 3.2a, the Wi-Fi face is the fastest responsive one, thus, all the Interests are sent over the Wi-Fi face.

- The *loadsharing* (LS) strategy *"distributes Interests amongst the available faces based on the unanswered queue size. This strategy operates most efficiently when there are multiple physical interfaces and the network is the limiting performance factor."* [16]. In the example shown in Fig. 3.2b, the Interests are distributed across the Wi-Fi and LTE faces.

- The *parallel* (PS) strategy *"sends Interests to all available faces in parallel. This strategy attempts to mask unstable links or poorly performing faces by redundantly sending Interests. This increases the overall network load and local processing overhead, and is not recommended when the links are of high quality"* [16]. In the example shown in Fig. 3.2c, each Interest is sent over the Wi-Fi and LTE faces in parallel.

For the evaluation of NetCodNDN, we always consider the *parallel* strategy, since by sending a single Interest over all its faces the client can receive multiple useful (*i.e.*, linearly independent) Data packets.

3.5.2 Metrics

To evaluate the performance of NetCodNDN, we measure the time $\Delta t_{measured}$ that a client needs in order to get the $|\hat{\mathscr{P}}_n|$ Data packets available at the sources. In the original NDN, $\Delta t_{measured}$ is defined as the elapsed time between the transmission of the first Interest and the reception of the $|\hat{\mathscr{P}}_n|$th missing Data packet. In NetCodNDN, $\Delta t_{measured}$ is defined as the elapsed time between the transmission of the first Interest and the reception of the $|\hat{\mathscr{P}}_n|$th linearly independent network coded Data packet, which permits to decode the whole generation of source Data packets. We consider that clients can have heterogeneous network resources. Thus, to make a fair comparison of the delivery delay, we define the *normalized delivery delay* as $d = \Delta t_{measured} / \Delta t_{min}$, where Δt_{min} is the theoretical lower bound on the time that a client would need in order to receive all the Data packets if it was alone in the network and was able to receive at max-flow rate. Thus, a normalized delivery delay equal to 1 means that the client was able to receive the complete set of Data packets at the

maximum rate. Note that $\Delta t_{measured} \geq \Delta t_{min}$, or equivalently, $d \geq 1$ always holds.

3.5.3 Butterfly Topology

We begin by evaluating NetCodNDN in the butterfly topology. We consider that every Data packet is stored randomly in at least one of the two sources, and a copy of the same Data packet is also placed in the CS of the other source with a replication probability $\phi \in [0,1]$. We set the bandwidth of every link in the network to $5Mbps$.

In the first set of experiments, we consider that $\phi = 1$, *i.e.*, both sources hold a copy of each Data packet in their CS. In this case, clients c_1 and c_2 can reach a copy of any Data packet over any of their faces. However, as explained in Section 3.2, with the original NDN architecture the maximum performance can be achieved only if both clients coordinate and send Interest messages for the same Data packets over the faces that connect them to the node r_4. In contrast, when network coding is employed, the need for coordination is eliminated, since clients do not send Interests for a specific Data packet but rather for any network coded Data packet.

Impact on Network Bottlenecks

Fig. 3.3 depicts the normalized delivery delay as a function of the bandwidth of the bottleneck link between the nodes r_3 and r_4. We can see that NetCodNDN achieves the optimal performance in the whole range of link bandwidth values evaluated. This is due to the fact that network coding removes the need for coordinating the forwarding of Interest messages. In contrast, the NDN forwarding strategies perform poorly and only the LS strategy can achieve the performance of NetCodNDN but it requires significantly higher link bandwidth. When the bottleneck link has the same bandwidth as all the other links, the average delivery time d of CCNx-LS is approximately 1.2 times the minimum delivery delay, Δt_{min}. This is caused by the randomness introduced by the LS strategy when choosing the faces over which Interests are transmitted when all the faces have the same load. This creates two extreme cases. In one case, all the Interests sent by both clients to node r_4 are the same, thus d tends to 1. In the other case, all the Interests are different, thus d tends to 1.33. This happens because each node receives 2/3 of the Data packets through the link connecting them to the sources, and 1/3 over the face connecting them to the node r_4, which means that 2/3 of the total packets travel on the bottleneck link. With the DS and the PS strategies,

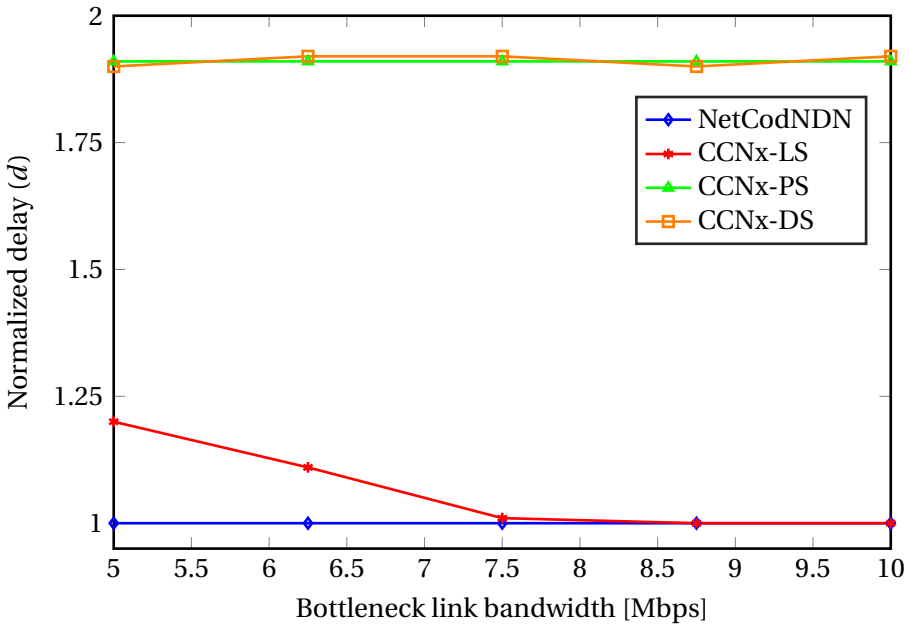

Figure 3.3: Normalized delivery delay vs. the bandwidth of the bottleneck link in the butterfly network.

the average delivery time is close to 2, as expected. With the DS strategy, the face connecting the clients to the sources is chosen as the best, and thus most of the Data packets are received over that face. With the PS strategy, each client forwards every Interest over both faces, thus bringing one copy of each Data packet over each face.

Pipeline Size

We now investigate how the number of concurrent Interests that a client can send, also known as the pipeline size, affects the performance in terms of the average normalized delivery delay. As shown in Fig. 3.4, the performance of NDN is optimized for a pipeline size value between 5 and 10, where the normalized delivery delay seen by the clients is 1.2. This is due to the fact that clients need to send at least 4 Interest messages over the faces connecting them to node r_4 in order to create a continuous flow of Data packets. Since the LS strategy distributes the Interests over all available faces, a client has to send 4 Interests over any face while it also has to send 3 or 4 Interests over the other face, which amounts to 7 or 8 Interests in total. For smaller pipeline sizes, the continuous flow is not set, while for larger pipeline sizes the number of Interests sent over the bottleneck link increases, thus worsening the client coordination problem. In contrast, the performance of NetCodNDN is not affected by the pipeline size, as it can

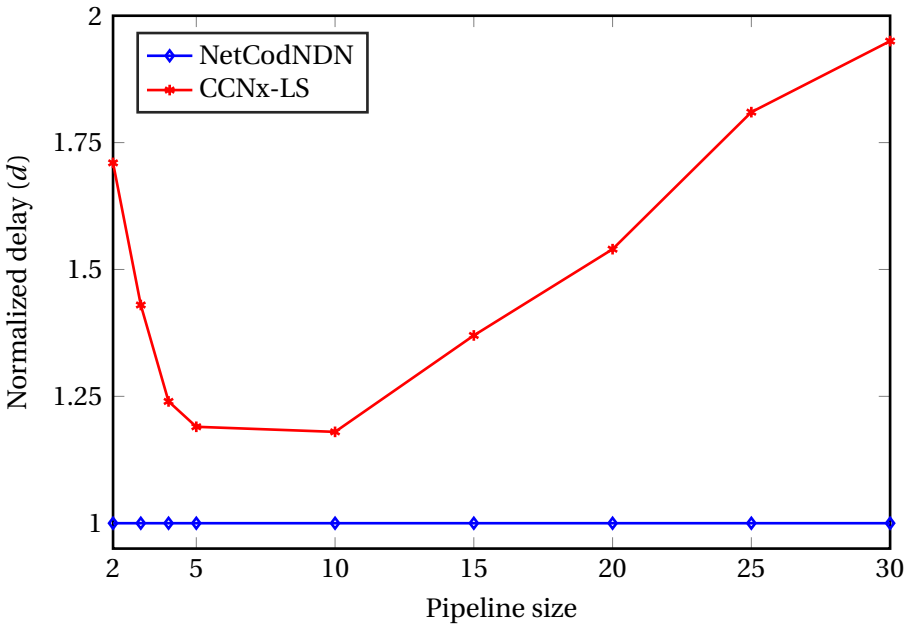

Figure 3.4: Normalized delivery delay vs. the pipeline size in the butterfly network.

be verified in Fig. 3.4. This can be explained by the fact that NetCodNDN eliminates the necessity that the clients request the same packets over the bottleneck link. For the rest of the experiments, without loss of generality, we choose a pipeline size of 10.

Impact of Packet Losses

In Fig. 3.5, we depict the influence of the packet loss rate on the performance of NetCodNDN and of the original NDN. We consider losses that are caused both by the transmission losses and the errors during the processing of the Interests and Data packets. We consider that the packet losses occur in an uniform manner, according to the desired packet loss rate. We can see that the performance of NDN with the LS strategy degrades faster than the performance of NetCodNDN as the packet loss rate increases. This is caused by the fact that in NDN, the client will be able to react to a packet loss only when the corresponding Interest expires, since any earlier retransmission of an Interest with the same prefix will be prevented by the PIT. Instead, with NetCodNDN, the clients can send Interests until they have a sufficient number of network coded Data packets in order to recover the content object. It is important to note that the maximum amount of concurrent Interests that a client can send is controlled by the pipeline size.

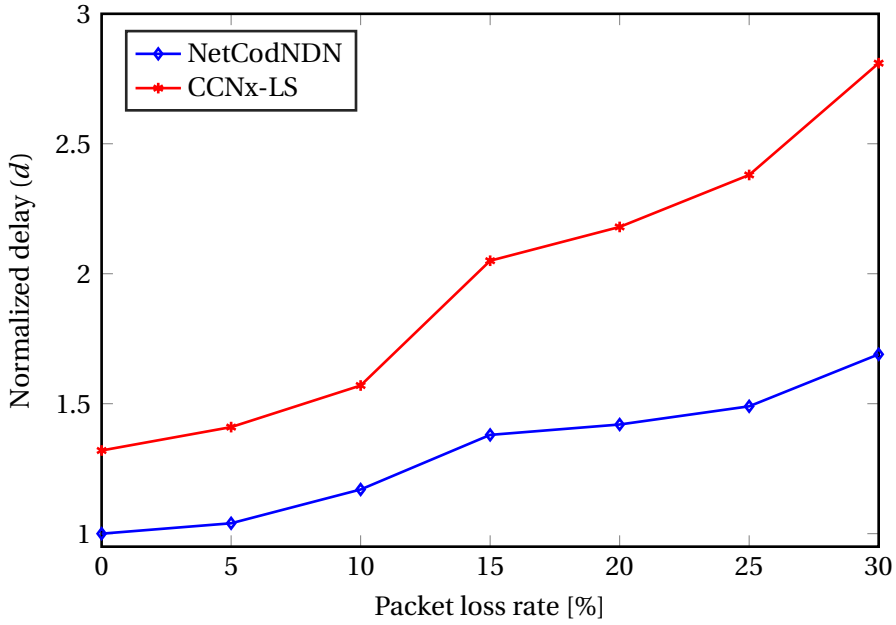

Figure 3.5: Normalized delivery delay vs. the packet loss rate in the butterfly network.

Source Replication

Finally, we evaluate the performance of NetCodNDN for different values of the replication probability ϕ. In CCNx, when $\phi < 1$, the clients should not only coordinate the requests sent over the bottleneck link as in the previous scenario, but they also should have the knowledge of the Data packets that each source stores, in order to avoid sending Interests over the face connecting them directly to the source that does not hold a copy of the requested Data packet. In Fig. 3.6, we can see that in NDN with the LS strategy, clients take 3.4 times longer to retrieve all the Data packets, when each Data packet is stored only in one of the sources. When the PS strategy is used, the clients do not need to know how the Data packets are distributed in the sources, since each Interest message is sent over both faces. However, since a copy of every Data packet crosses the bottleneck link, the traffic over the bottleneck link is doubled compared to the network coding case. When the probability that the Data packets are stored in both sources increases, the performance of CCNx with the LS strategy improves, but eventually saturates at 1.2 times the minimum delay, which is consistent with the results depicted in Fig. 3.4.

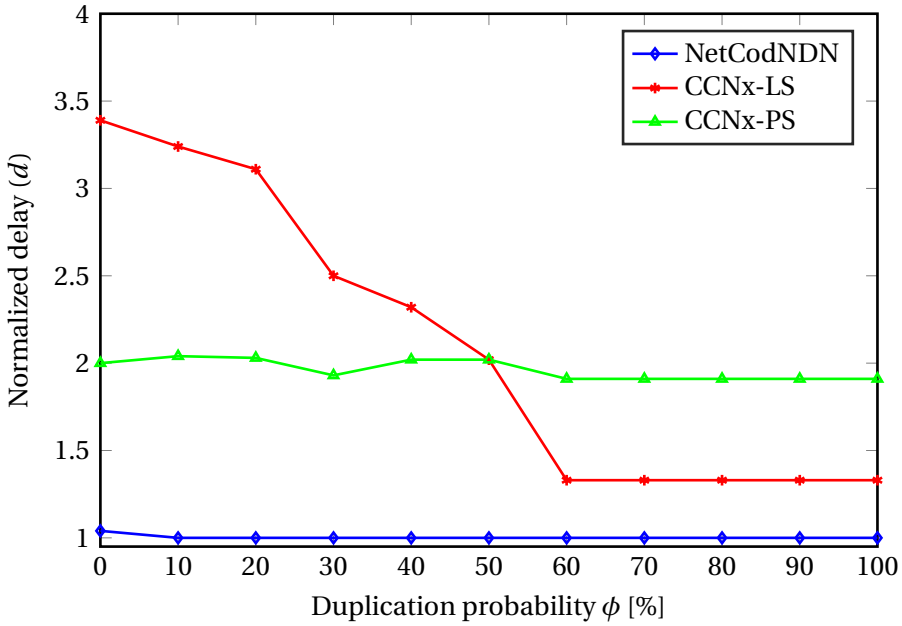

Figure 3.6: Normalized delivery delay vs. source Data replication probability in the butterfly network.

3.5.4 PlanetLab Topologies

We now evaluate our architecture in more realistic network topologies captured by the PlanetLab project [72]. We use the network topology shown in Fig. 3.7 that consists of one source node, 5 client nodes and 20 routers. The links connecting the nodes have a capacity of 12Mbps. The topology is generated in the following way. First, a random source is chosen. Then, the routers are randomly added one-by-one to the topology, taking into consideration the number of desired routers. Finally, the clients are connected to the edge routers. Since the selection of routers is randomized, the routers that cannot be reached by the source and the routers that are not connected with any client are removed. This procedure is better described in [22]. We measure the normalized delivery delay d for each client and then compare its average.

Number of Clients

First, we investigate how the performance is affected by the number of clients in the network. In Fig. 3.8, we can see that with a single client (in this case node 24), NetCod-NDN and CCNx perform similarly. In this case, network coding does not introduce any gains since there is only one client in the network and no losses are considered.

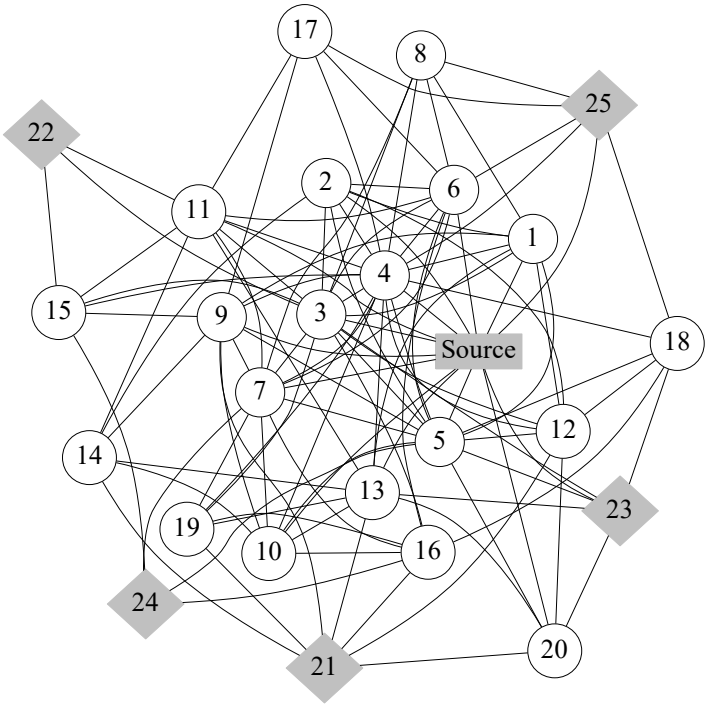

Figure 3.7: PlanetLab topology used.

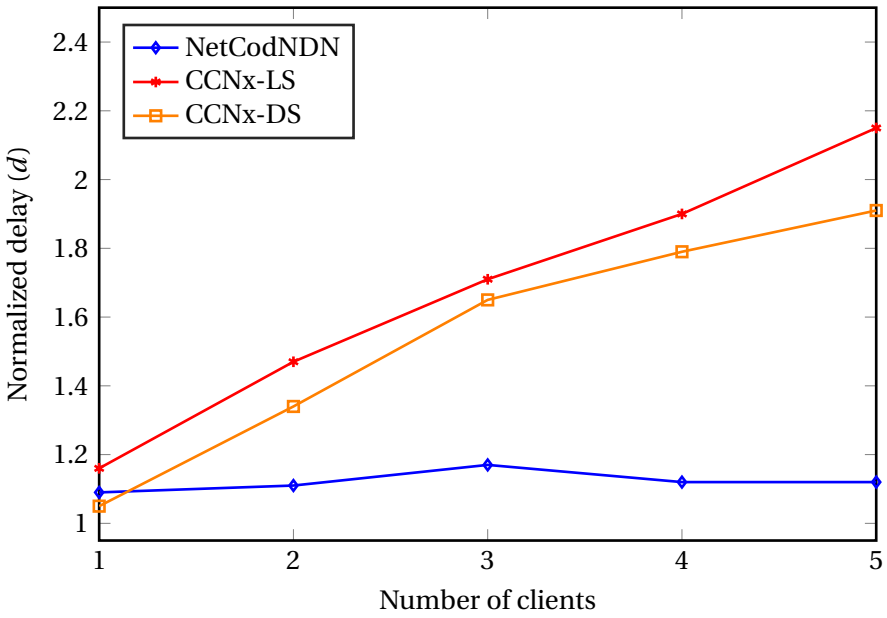

Figure 3.8: Normalized delivery delay vs. the number of clients in the network in the PlanetLab topology.

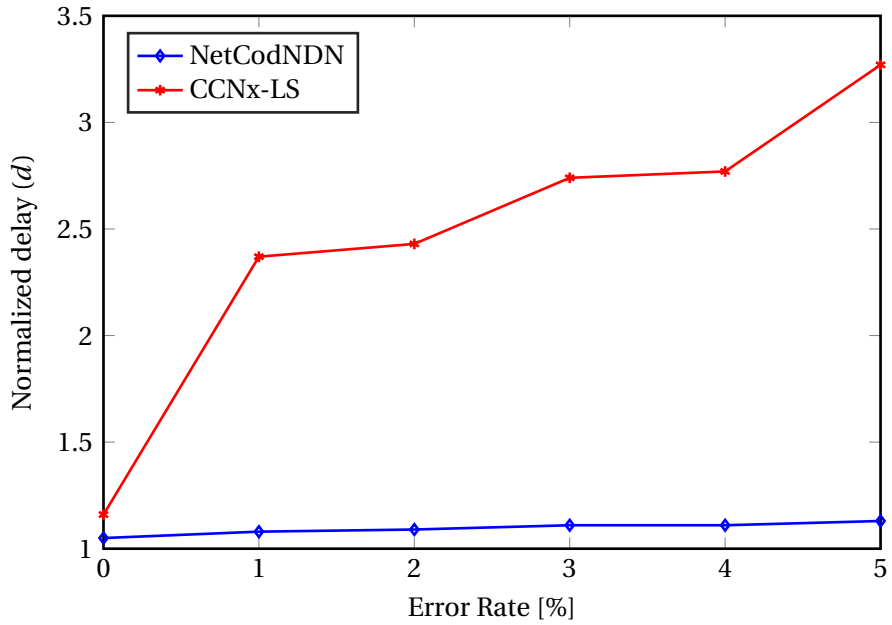

Figure 3.9: Normalized delivery delay vs. the packet transmission error rate in the PlanetLab topology.

However, the performance of CCNx starts to degrade with the introduction of more clients, as they start to compete for the network resources. In contrast, we can see that the performance of NetCodNDN does not degrade with the addition of new clients to the network topology. These results show that the NetCodNDN architecture uses more efficiently the available network resources.

Impact of Packet Losses

We also evaluate how the packet losses in the network affect the performance of the NetCodNDN for larger topologies. For this evaluation, we choose to keep only one client, in order to compare the results with the performance of the NDN. As with the butterfly topology, we consider packet losses that are caused both by transmission losses and errors introduced during the processing of the packets. In Fig. 3.9, we can see that NetCodNDN maintains the delivery delay close to the expected one, while the performance of NDN degrades very fast with the introduction of packet losses. As in the butterfly topology, this fast degradation is due to the fact that when a packet is lost, the client needs to wait until the corresponding Interest expires before it can re-send a new one.

3.6 Conclusions

In this Chapter, we have presented NetCodNDN, an architecture that integrates network coding into NDN. In NetCodNDN, the clients send Interests for network coded Data packets with a given name prefix, instead of asking for specific Data packets as in NDN. The network nodes combine the Data packets by means of Random Linear Network Coding (RLNC) before forwarding them, to take advantage of the benefits that network coding brings to the data retrieval process. Our architecture is able to *(i)* simplify the aggregation of Interests for network coded content; *(ii)* reduce the number of duplicate Data packets; and *(iii)* allow clients to send multiple Interests for the same content in parallel. The NetCodNDN architecture has been tested in networks with multiple clients and sources, where we have observed large performance gains in terms of the time needed to retrieve the demanded content. In the next Chapter, we use the NetCodNDN architecture proposed in this Chapter to demonstrate and quantify the benefits that network coding brings to video streaming, a particular data intensive application, over NDN.

4

Adaptive Video Streaming over Network Coding Enabled NDN

4.1 Introduction

This Chapter studies and quantifies the benefits that network coding brings to data intensive applications in NDN, using the NetCodNDN architecture proposed in Chapter 3.

Since the volume of video traffic on the Internet is a large proportion of all the traffic on the Internet [21], in this Chapter we consider video streaming as a data intensive application that can benefit from the use of network coding in the NDN architecture.

One of the most prominent video streaming techniques used nowadays is adaptive video streaming, and, in particular, Dynamic Adaptive streaming over HTTP (DASH) [37]. One of the main characteristics of DASH is that the clients are in control of the streaming logic, deciding the bitrate and resolution of the streamed video. The client driven video retrieval and adaptation property in DASH resembles the content

retrieval mechanism of NDN and NetCodNDN. In particular, NDN, NetCodNDN and DASH are all client driven and, hence, clients request content by sending requests with the names assigned to each piece of content, *i.e.*, a content object in NDN and NetCodNDN, or a video segment in DASH.

Motivated by the similarities of content retrieval mechanisms of NetCodNDN and DASH, in this Chapter we propose a Dynamic Adaptive Streaming over NetCodNDN (DAS-NetCodNDN) architecture [80], which brings the benefits of network coding to NDN video streaming applications. We propose a new model of a client and a source that enable dynamic adaptive streaming of video over network coding enabled NDN. Moreover, we have updated the design of the NetCodNDN architecture presented in Chapter 3, to make it more efficient for data intensive applications. In particular, we have redesigned the Content Store (CS) and the Pending Interest Table (PIT), which now are able to handle a high number of entries in more efficient way.

The updated NetCodNDN architecture has been implemented on top of the ndnSIM [54] codebase, using the original NDN project codebase [61]. Then, the DAS-NetCodNDN has been implemented by appropriately modifying the updated NetCodNDN codebase. We compare the video streaming performance of DAS-NetCodNDN to the original NDN architecture in a Netflix-like scenario, designed with parameters available in the literature [2, 12, 62]. Our results demonstrate that by using network coding, DAS-NetCodNDN exploits more effectively the multipath communication and attains a higher cache hit rate in the routers. This translates into lower bandwidth consumption at the sources, as well as higher bitrate seen at the clients. As a result, clients can reach their desired video quality faster.

This Chapter is organized as follows. We start by describing how the network coded Data packets are formed, using DASH [37] and NetCodNDN segmentation schemes. Then, we present the design of the end-point applications, *i.e.*, the clients and the sources, which run at the application layer of the network. After that, we introduce a set of optimizations to the content store and the pending Interest table of the NetCodNDN architecture that render it more efficient when processing Interests and Data packets. Finally, we present the evaluation of the DAS-NetCodNDN architecture, showing the benefits that network coding brings to both end-users and video content providers in a video streaming scenario.

4.2 The DAS-NetCodNDN Architecture

In this section, we present DAS-NetCodNDN, an architecture for dynamic adaptive streaming (DAS) over network coding enabled NDN (NetCodNDN). Our architecture is based on the NetCodNDN architecture proposed in Chapter 3, which here is advanced to support dynamic adaptive video streaming. We start by defining the video segmentation and naming scheme in the proposed architecture. Then, we describe a new set of changes to the data structures of a NetCodNDN node, *i.e.*, the Content Store (CS) and the Pending Interest Table (PIT), which improve the processing of network coded Data packets and Interests.

4.2.1 Video Fragmentation

We consider a set of videos $\mathcal{V}$ that are made available by a *video content provider* to a set of *end-users*. One of the main characteristics of adaptive video streaming is that the clients are in control of the streaming logic, deciding the bitrate and resolution of the streamed video. To enable the video quality adaptation by the clients, each video $v \in \mathcal{V}$ is encoded with different parameters (*e.g.*, bitrate, resolution, *etc.*), creating a set of *representations* $\mathcal{Q}$. The video data in each representation $q \in \mathcal{Q}$ is divided into a set of *segments* $\mathcal{Z}$. Every segment $z \in \mathcal{Z}$ has the same duration. This allows clients to request segments belonging to the representation that better adapts to their current network conditions, display capabilities, *etc.* Hence, the clients can switch to a different representation after receiving each segment if their conditions change in between, *e.g.*, an increase or decrease in the available bandwidth, a change of the screen conditions, *etc.*. Each particular video segment is identified with a name $n = \texttt{v/q/z}$ that is composed of the ID v of the video to which the segment belongs, the representation q in which the segment has been encoded and the segment ID z.

Since the size of the video segments is usually larger than the Maximum Transmission Unit (MTU), the DASH segments are divided into smaller Data packets that fit into an MTU. Therefore, we consider that a DASH segment with name $n = \texttt{v/q/z}$ is composed of the set of Data packets $\mathcal{P}_n = \{p_{n,1}, \ldots, p_{n,|\mathcal{P}_n|}\}$, similarly to the data segmentation scheme presented in Chapter 3.

To inform the clients about the offered video representations, a file called the Media Presentation Description (MPD) is associated with each video v. This file contains information about the available representations $\mathcal{Q}$ in which the video v has been

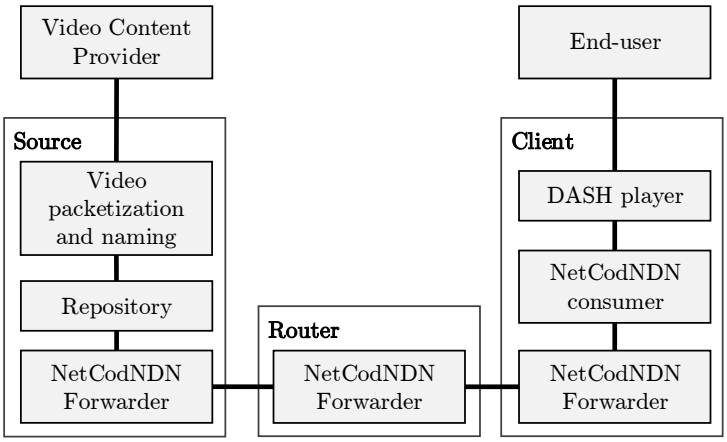

Figure 4.1: Proposed architecture for adaptive video streaming over network coding enabled NDN.

encoded and the segments $\mathscr{Z}$ that compose each representation, among other parameters. A client that is interested in receiving the video v should first request the MPD file associated with this video. Then, after receiving and parsing the MPD file of the video v, the client knows what representations are available and what names it should use to request the video segments.

4.2.2 Adaptive Video Streaming Implementation

In Fig. 4.1, we show a simple network example composed of a source $s \in \mathscr{S}$, a router $r \in \mathscr{R}$, and a client $c \in \mathscr{C}$, and we illustrate the main components of these nodes. In the following sections, we discuss the source and client node implementations in more details.

Source nodes

We consider that a source $s \in \mathscr{S}$ is a node that persistently stores the Data packets that form the video segments. In particular, a source s stores the set of Data packets $\mathscr{P}^s = \bigcup_{n,g} \mathscr{P}^s_{n,g}$. It is worth noting that a source s may not store the whole set of Data packets $\mathscr{P}_{n,g}$, but only a subset $\mathscr{P}^s_{n,g} \subset \mathscr{P}_{n,g}$.

In order to accomplish this function, each source s contains one or more *repositories* where the Data packets are stored. Repositories and content stores have similar functionalities: *(i)* when they receive a Data packet $p_{n,g} \in \mathscr{P}_{n,g}$, they store it in their memory; and *(ii)* when they receive an Interest $\hat{i}_{n,g}$, they return a network coded Data

packet $\hat{p}_{n,g}$ generated with the set of Data packets $\mathscr{P}^{s}_{n,g}$ that are available in their memory. However, differently from a CS, a repository is designed to keep the Data packets for longer time periods, using persistent storage devices.

Before the video segments are requested by the clients, the sources are initialized following three steps: *(i)* the source receives from a content provider a set of videos that it should store, then, *(ii)* the videos are divided into Data packets and generations and given a name, as described in Section 4.2.1, and finally, *(iii)* the Data packets are loaded into the repository. Then, the repository replies with network coded versions of these Data packets whenever it receives an Interest that matches the name prefix.

Client nodes

In our architecture, we consider that a client $c \in \mathscr{C}$ is a node that, upon receiving a video request from an end-user, generates the Interests needed to collect its Data packets and assembles the video segments that will be delivered to the end-user. Our model of a client consists of two main applications: *(i)* an adaptive video player that decides the video representation that should be displayed and requests the appropriate segments, and *(ii)* a NetCodNDN consumer that receives requests for segments and generates Interests for network coded Data packets. Moreover, when the network coded Data packets arrive at the NetCodNDN consumer, it decodes them and reassembles the original segment before sending it to the adaptive video player. These components are further described below.

The *adaptive video player* is the most direct interface between the end-user and the video communication system. When an adaptive video player receives a request from an end-user to retrieve a video v, it first requests the MPD file. This file is typically of a small size, *i.e.*, it usually fits into a small number of Data packets. For this reason, applying network coding to it will not bring noticeable benefits, and thus, in our architecture it is requested as traditional NDN content object. Given the MPD information, the available resources and the configuration parameters, the adaptive video player decides which representation $q \in \mathscr{Q}$ is the optimal one every time that it has to request a new segment $z \in \mathscr{Z}$. The adaptive video player is agnostic to the protocol deployed in the network for requesting the segment, which can be either NetCodNDN, NDN or HTTP. It only takes into account the incoming bandwidth measured in the reception of the previous video segments.

When the adaptive video player decides to request a video segment with name n = `v/q/z`, it sends this request to the *NetCodNDN consumer*, which then generates and forwards a set of Interests $\hat{\mathscr{I}}_n$ that request the set of packets $\mathscr{P}_n$ that form the video segment. Whenever the NetCodNDN consumer collects the complete set of Data packets $\mathscr{P}_n$ that form the requested segment, it assembles the segment and passes it to the adaptive video player.

Since the network coded Data packets are grouped into generations, the set of Interests $\hat{\mathscr{I}}_n$ is also divided in generations, thus $\hat{\mathscr{I}}_n = \bigcup_{g=1}^{G} \hat{\mathscr{I}}_{n,g}$, where g is the generation ID and G is the total number of generations. The total number of generations G can be obtained as Data packet metadata, or simply by adding a flag to the last generation. These Interests are passed to the network layer, that forwards them to the node's neighbors. The generations are requested sequentially, starting from $g = 1$ up to the last generation, G. The client only starts to send the Interests $\hat{\mathscr{I}}_{n,g}$ when all the Data packets $\mathscr{P}_{n,g-1}$ from the previous generation have been received. The total number of generations G can be obtained as Data packet metadata, or simply by adding a flag to the Data packets of the last generation. The NetCodNDN consumer keeps track of the received innovative Data packets $\hat{\mathbf{p}}_{n,g}$ in a matrix $\hat{\mathbf{P}}^{c}_{n,g}$, where c stands for the consumer, so that the original set of Data packets can be retrieved by performing Gaussian elimination when the matrix $\hat{\mathbf{P}}^{c}_{n,g}$ is of full rank, *i.e.*, it contains $|\mathscr{P}_{n,g}|$ linearly independent Data packets.

4.2.3 Improvements to the NetCodNDN Architecture

While developing and testing DAS-NetCodNDN, we have noticed that the NetCodNDN architecture explained in Chapter 3 did not scale well with the number of Interests and Data packets required by data intensive applications, *e.g.*, video streaming. In this section we present a set of changes to the NetCodNDN architecture that make it scalable for adaptive video streaming. In particular, we have changed the Content Store (CS) and the Pending Interest Table (PIT) data structures to deal with the large number of Interests and Data packets required by data intensive applications, *e.g.*, video streaming. Note that the NetCodNDN architecture explained in Chapter 3 utilized the original CS and introduced a few modifications to the PIT, keeping the model of the proposed NetCodNDN as similar as possible to the model of NDN [38, 104], at the expense of performance. The new CS and PIT models presented in this section are redesigned to improve the performance of the network coding operations

in data intensive scenarios.

Also note that in our architecture, other than the redesigned CS and PIT, the nodes still have the traditional CS and PIT to process non-network coding Interests and Data packets.

Content Store

The NetCodNDN architecture presented in Chapter 3 uses the CS model provided by NDN [38, 104] for both the traditional Interests and Interests for network coded data. In this case, generating a Data packet $\hat{p}_{n,g}$ requires up to $|\hat{\mathbf{P}}_{n,g}|$ lookups to the CS, since each Data packet is stored as an independent CS entry. Differently, in this Chapter we propose a new design of the NetCodNDN CS that reduces the complexity of the creation of new network coded Data packets, with respect to that presented in Chapter 3. In the redesigned NetCodNDN CS, each CS entry contains a set of network coded Data packets, $\hat{\mathscr{P}}^{v}_{n,g}$, where all the Data packets belong to the same generation g. This set of Data packets is stored as a matrix $\hat{\mathbf{P}}^{v}_{n,g}$, where each row is a vector $\hat{\mathbf{p}}_{n,g}$ that represents the network coded Data packet $\hat{p}_{n,g}$. This allows reducing the number of lookups to the CS needed to generate a network coded Data packet to only one, which is much more efficient than the CS proposed in Chapter 3.

Moreover, in the redesigned CS, each CS entry also stores a counter $\sigma^{f}_{n,g}$ for each face f of the node v. Each counter $\sigma^{f}_{n,g}$ measures the number of Data packets generated with the matrix $\hat{\mathbf{P}}^{v}_{n,g}$ that have already been sent over face f, *i.e.*, it measures the amount of information from the matrix $\hat{\mathbf{P}}^{v}_{n,g}$ that has been transferred from the node v to the neighbor node connected over face f. When a Data packet with name prefix (n,g) is removed from $\hat{\mathbf{P}}^{v}_{n,g}$ (*e.g.*, when the CS replacement policy decides that a Data packet with name prefix (n,g) needs to be removed from the CS), the amount of information in $\hat{\mathbf{P}}^{v}_{n,g}$ is reduced by 1. Therefore, the value of $\sigma^{f}_{n,g}$ is also decreased by 1 for all the faces, in order to reflect the current state of the CS in terms of the available information.

The comparison of the structures of the NDN and NetCodNDN CSs is shown in Fig. 4.2. In this example, both CSs are storing three Data packets with name prefix n. In the NDN CS, there are three CS entries, each one storing a single Data packet, as it can be seen in Fig. 4.2a. Differently, as it can be seen in Fig. 4.2b, in the NetCodNDN CS there is a single entry that contains a matrix $\hat{\mathbf{P}}^{v}_{n,g}$ that stores the content of the three Data

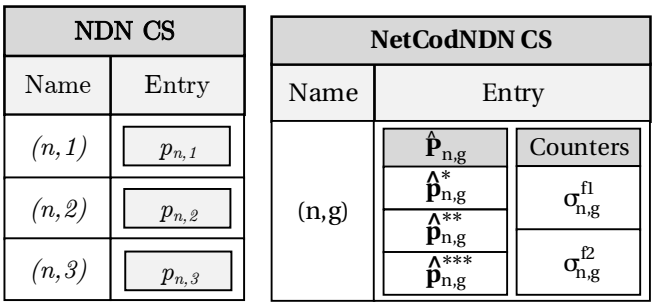

Figure 4.2: Comparison of (a) the CS presented in Chapter 3 vs. (b) the redesigned CS presented in this Chapter.

packets, and a set of counters $\sigma^{f}_{n,g} \forall f$ that store the number of Data packets that have already been sent over each face.

It is worth noting that not every NetCodNDN router should keep a CS with the Data packets that it receives. As demonstrated by Fayazbakhsh *et al.* [26] and Sun *et al.* [87], the content delivery performance of an NDN network in which every node has a CS is not much better than that of an NDN network in which only the edge routers have a CS, taking into consideration the computing power and storage capacity that a CS requires. It is also worth noting that not all the NetCodNDN routers need to apply network coding operations to the received Data packets. In fact, when a NetCodNDN node does not have a CS, it will not be able to apply network coding operations on the Data packets before forwarding them, since there will be no other Data packets cached in the node. If the nodes that are able to apply network coding are chosen carefully, the benefits that network coding brings to the video delivery remain high, while the overall computing power and storage capacity of the network can be drastically reduced, as has been demonstrated by Cleju *et al.* [22].

Pending Interest Table

To improve the functionalities of the NetCodNDN forwarder in the presence of a large number of Interests, we have redesigned the PIT in-record with respect to that presented in Chapter 3.

The redesigned NetCodNDN PIT's in-record $t^{f(in)}_{n,g}$ is a list that keeps track of the Interests $\hat{i}_{n,g}$ that arrived over face f. Each element in this list is a tuple of the form $t^{f,\rho(in)}_{n,g} = (\rho, e)$, where ρ is the rank that the matrix $\hat{\mathbf{P}}^{\nu}_{n,g}$ must have before replying to

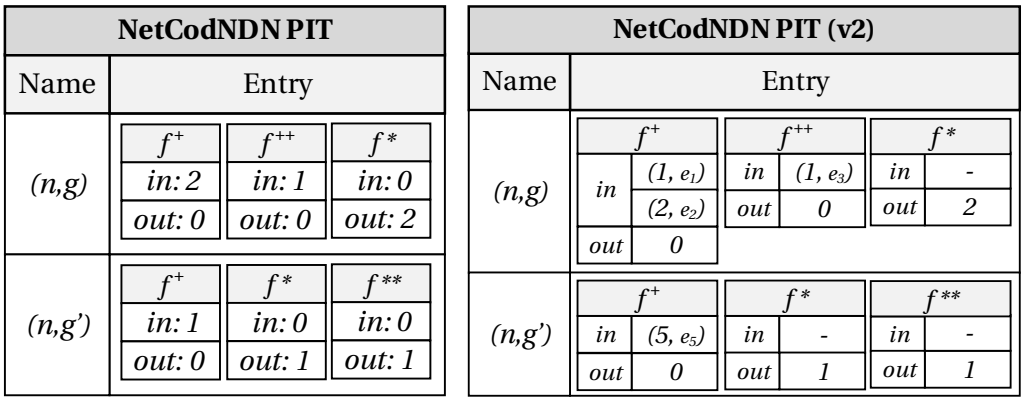

Figure 4.3: Comparison of (a) the original NetCodNDN PIT vs. (b) the redesigned NetCodNDN PIT.

the Interest, and e is the expiration time of the Interest. The size of this list, denoted as $|t_{n,g}^{f(in)}|$, is the total number of Data packets with name prefix (n,g) that should be sent over face f.

The comparison of the structures of the original NetCodNDN and the redesigned NetCodNDN PITs is shown in Fig. 4.3. In this example, both PITs store two pending Interests with name prefix n. As it can be seen in Fig. 4.3b, the redesigned NetCodNDN PIT stores information about the rank at which each Interest should be satisfied, together with the expiration time of each Interest. This information is not available in the original NetCodNDN PIT in-record, as it can be seen in Fig 4.3a.

The redesigned NetCodNDN PIT provides methods to insert information about new Interests, and to get the PIT entry associated with a particular name prefix. In particular, the following two methods are provided:

- `GetPIT((`n,g`))` – Gets the PIT entry $t_{n,g}$ associated with the name prefix (n,g), if it exists.

- `InsertInPIT((`n,g`),`f`,`e`)` – Inserts a new record $(\rho+1,e)$ into the in-record $t_{n,g}^{f(in)}$ of the PIT entry $t_{n,g}$ associated with the name prefix (n,g), where ρ is the highest rank that is currently pending in the in-record $t_{n,g}^{f(in)}$. If the PIT entry does not exist, it is created by this method.

- `RemoveInPIT((`n,g`),`f`,`ρ`)` – Removes the record associated with the rank ρ

from the in-record $t_{n,g}^{f(in)}$ of the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, this method does nothing.

- `InsertOutPIT(`(n,g)`,`f`)` – Increases by 1 the value of $t_{n,g}^{f(out)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, it is created by this method.

- `RemoveOutPIT(`(n,g)`,`f`)` – Decreases by 1 the value of $t_{n,g}^{f(out)}$ in the PIT entry $t_{n,g}$ associated with the name prefix (n,g). If the PIT entry does not exist, this method does nothing.

Interest Processing

In the previous Sections we have presented a series of changes to the NetCodNDN CS and PIT tables that improve the performance of the network coding operations in data intensive scenarios. As a consequence of the proposed changes to the CS and PIT, new procedures to process the Interests at the NetCodNDN nodes. The changes to the Interest processing procedure that has been described in Section 3.4.4 are further explained below and summarized in Algorithm 3.

Replying to an Interest — This part of the Interest processing procedure (lines 1 - 8) is similar to the one defined in Section 3.4.4 of Chapter 3.

Forwarding an Interest — When the number of network coded Data packets that can be generated by node v and that have a high probability to be innovative, $\xi_{n,g}^{f}$, is equal to 0 (line 9), node v adds a new tuple $(\rho+1, e)$ to the in-record $t_{n,g}^{f(in)}$, where ρ is the highest rank on the in-record and e is the expiration time of the Interest $\hat{i}_{n,g}$ (line 11). Then, node v computes the number $\epsilon_{n,g}$ of innovative network coded Data packets with name prefix (n,g) that it is expecting to receive before the Interest $\hat{i}_{n,g}$ expires. The value of $\epsilon_{n,g}$ is computed as the total number of Interests with name prefix (n,g) that have been forwarded by the node v over all its faces, *i.e.*, the sum of all the out-records in the PIT entry $t_{n,g}$ (line 13). Finally, the node forwards the Interest $\hat{i}_{n,g}$ if the number of innovative network coded Data packets with name prefix (n,g) that it is expecting to receive is less than or equal to the number of Data packets with name prefix (n,g) that are pending to be sent over face f, *i.e.*, $\epsilon_{n,g} \leq t_{n,g}^{f(in)}$ (lines 14 - 16).

Waiting for a new network coded Data packet — This part of the Interest processing

Algorithm 3 Redesigned Interest processing in the NetCodNDN forwarder

Require: $\hat{i}_{n,g}, f, \mathbf{P}^{v}_{n,g} \leftarrow$ `GetCS(`n,g`)`

1: **if** `rank(`$\hat{\mathbf{P}}^{v}_{n,g}$`)` $= |\hat{\mathscr{P}}_{n,g}|$ **then** (*Generation is decodable*)
2: $\quad \xi^{f}_{n,g} = |\hat{\mathscr{P}}_{n,g}|$
3: **else**
4: $\quad \xi^{f}_{n,g} =$ `rank(`$\hat{\mathbf{P}}^{v}_{n,g}$`)` $- \sigma^{f}_{n,g}$
5: **end if**
6: **if** $\xi^{f}_{n,g} > 0$ **then**
7: $\quad \hat{\mathbf{p}}_{n,g} \leftarrow \sum_{j=1}^{|\hat{\mathbf{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
8: $\quad$ Send Data packet $\hat{p}_{n,g}$ over face f
9: **else**
10: $\quad e \leftarrow$ expire time in the Interest $\hat{i}_{n,g}$)
11: $\quad$ `InsertInPIT ((`n,g`),`f`,`e`)`
12: $\quad t_{n,g} \leftarrow$ `GetPIT ((`n,g`))`
13: $\quad \epsilon_{n,g} = \sum_{f'}^{\mathscr{F}^{v}_{n,g}} t^{f'(out)}_{n,g}$ (*Total number of Interests $\hat{i}_{n,g}$ forwarded*)
14: $\quad$ **if** $\epsilon_{n,g} \leq t^{f(in)}_{n,g}$ **then**
15: $\quad\quad$ Forward the Interest $\hat{i}_{n,g}$ over face f_{out}.
16: $\quad\quad$ `InsertOutPIT ((`n,g`),`f_{out}`)`
17: $\quad$ **else**
18: $\quad\quad$ Wait for new Data packets.
19: $\quad$ **end if**
20: **end if**

procedure (lines 18 - 19) is also similar to the one defined in Section 3.4.4 of Chapter 3.

Data Packet Processing

The Data packet processing procedure has also been changed to adapt it to the redesigned PIT and CS. The changes are described below and are outlined in Algorithm 4.

Differently from the Data packet processing procedure that has been described in Section 3.4.5, in the new procedure node v does not need to care about multiple appearances of the same face in the PIT entry, since they are organized by rank ρ in

Algorithm 4 Data packet processing in the NetCodNDN forwarder

Require: $\hat{p}_{n,g}$, f

1: $t_{n,g} \leftarrow$ `GetPIT ((`n,g`))`
2: **if** $t_{n,g} = \varnothing$ **then** (*Unsolicited*)
3: Discard $\hat{p}_{n,g}$
4: **else**
5: `RemoveOutPIT((`n,g`),`f`)`
6: **if** `rank(`$\hat{\mathbf{P}}^{v}_{n,g} \cup \hat{p}_{n,g}$`) >` `rank(`$\hat{\mathbf{P}}^{v}_{n,g}$`)` **then**
7: `InsertCS (`$\hat{p}_{n,g}$`)`
8: $\rho \leftarrow$ `rank(`$\hat{\mathbf{P}}^{r}_{n,g}$`)`
9: **for all** $f' \in t_{n,g}$ **do**
10: **if** ρ exists in $t^{f'(in)}_{n,g}$ and it is not expired **then**
11: $\hat{p}^{*}_{n,g} = \sum_{j=1}^{|\hat{\mathscr{P}}^{v}_{n,g}|} a_j \cdot \hat{\mathbf{p}}^{(j)}_{n,g}$
12: Send the Data packet $\hat{p}^{*}_{n,g}$ over face f'
13: $\sigma^{f}_{n,g} \leftarrow \sigma^{f}_{n,g} + 1$
14: `RemoveInPIT((`n,g`),`f`,`ρ`)`
15: **end if**
16: **end for**
17: **else**
18: Discard $\hat{p}_{n,g}$
19: **end if**
20: **end if**

the in-record. Thus, in the new procedure, node v forwards a Data packet to all the faces where a non-expired pending Interest exists for the current rank ρ of the matrix $\hat{\mathbf{P}}^{r}_{n,g}$ in the PIT in-record (line 10).

4.3 Evaluation

In this section, we evaluate the performance of our proposed DAS-NetCodNDN architecture, and use it to show the benefits that network coding bring to data intensive applications, in particular to video streaming. We compare the performance of

DAS-NetCodNDN with that of an NDN variant without network coding capabilities (DAS-NDN). In the following sections we first describe the implementation of our architecture, the network topology used in the experiments, and the evaluation setup. Then, we show the performance evaluation results of our proposed adaptive video streaming architecture.

4.3.1 Implementation

We implemented our proposed DAS-NetCodNDN by extending both the NDN layer, to enable the NetCodNDN forwarder at every node, and the application layer, to enable adaptive video streaming at the clients and the sources.

- *NetCodNDN forwarder* — The NetCodNDN forwarder is implemented by integrating the changes to the NDN architecture described in Chapter 3 and the improvements described in Section 4.2.3 into the NDN Forwarding Daemon (NFD) codebase [61]. We have modified two main modules of the NFD code to implement NetCodNDN. First, we have modified the module that implements the NDN router tables, where two new tables were implemented: the modified CS and PIT, as described in Sections 4.2.3 and 4.2.3, respectively. In the NetCodNDN codebase both the original and the modified versions of the CS and PIT coexist. The original version is used to process NDN Interests and Data packets, while the modified version is used in the processing of network coding Interests and Data packets. We have also modified the *Forwarding* module, in order to add a new set of methods to process network coding enabled Interests and Data packets. The modified Forwarding module uses the original NDN procedures [4] to process traditional Interests and Data packets. Further, it uses the NetCodNDN procedures, described in Chapter 3, to process the network coding Interests and Data packets. We have used the Kodo C++ library [69] to enable network coding operations in the NetCodNDN forwarder.

- *Adaptive video applications* — To implement the sources and clients that enable adaptive video streaming with network coding, as described in Section 4.2.2, we modified the Adaptive Multimedia Streaming with ndnSIM (AMuSt) codebase [43]. The AMuSt framework provides a set of applications for producing and consuming adaptive video, based on the DASH standard [37], but replacing HTTP with NDN. The DASH functionality is provided by the libdash library [60],

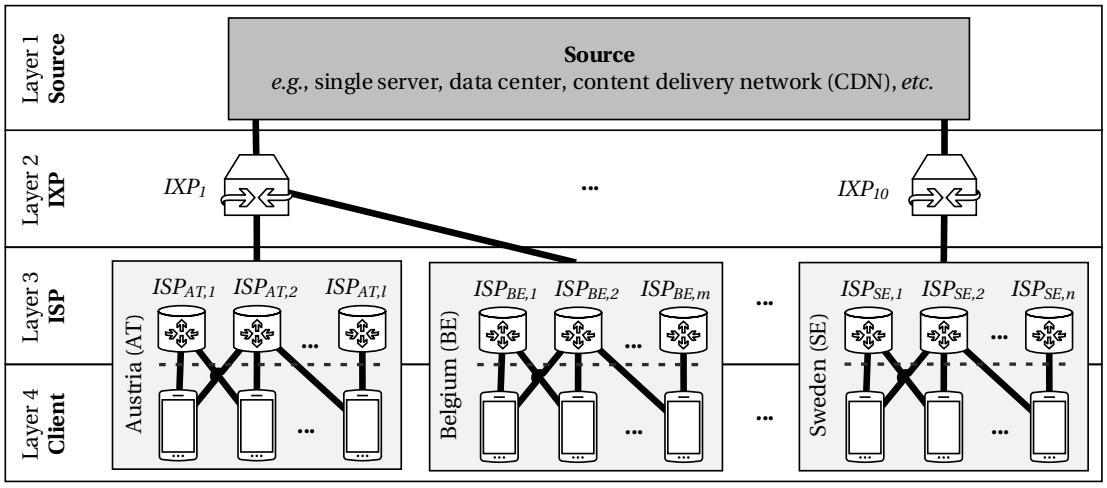

Figure 4.4: Layered topology used in the evaluation.

an open-source library that provides an interface to the DASH standard. Currently, libdash is the official reference software of the DASH standard. We implemented a new set of applications in the AMuSt framework that use the Kodo C++ library [69] to enable network coding at both the sources and the clients.

Finally, we have installed the implementations of the NetCodNDN forwarder and the adaptive video applications into ndnSIM [54] nodes. ndnSIM is an NDN simulator based on the NS-3 network simulator [63]. This simulator is used to generate the network nodes, *i.e.*, sources, routers and clients, and connect them with point-to-point links.

4.3.2 Network Topology

We evaluate our proposed DAS-NetCodNDN in a layered topology, presented in Fig. 4.4. The design of this topology is inspired by Netflix's OpenConnect Content Delivery Network [12], and it is composed of four layers.

The first layer of our topology contains a source that is able to provide all the video segments that the clients might request. The source can be considered as a server that stores all the video segments or as a connection to a Content Delivery Network (CDN) that is able to provide the video segments from any of its servers.

The second layer of our topology contains a set of routers that represent Internet ex-

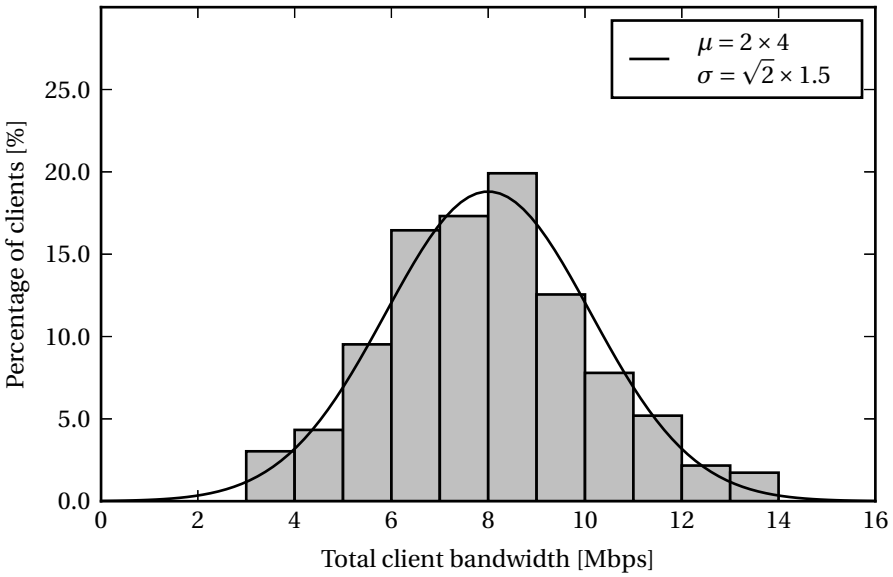

Figure 4.5: Clients' bandwidth distribution. Each client has two faces, the average bandwidth is $\mu = 2 \times 4$Mbps and the standard deviation is $\sigma = \sqrt{2} \times 1.5$.

change points (IXP). In our evaluation, we consider 10 IXP routers, each one connected directly to the source.

The third layer of the topology is another set of routers that represent Internet service provider (ISP) routers. Each ISP router is connected to two or three IXP routers. Moreover, the ISP routers are clustered into 16 groups that represent the European countries served by Netflix [12]. Netflix deploys video delivery servers in certain IXPs and ISPs. These servers store the complete or a fraction of the Netflix video catalog. In our topology, the IXP and ISP routers have content stores that can cache all the incoming Data packets. When cache space is limited, our architecture needs a cache replacement strategy to decide the cached content. This is, however, out of the scope of this Chapter which aims to study the behavior of the proposed adaptive video streaming protocol. The study of such caching strategies is presented in Chapter 5. Each country (*i.e.*, cluster) has between 4 and 10 ISP routers.

Finally, the fourth layer of the topology consists of a set of clients, each of them is connected to two ISP routers that belong to the same country as the client. Each country has between 10 and 20 clients. We choose to connect each client with two ISP routers to evaluate multi-path adaptive video streaming, considering that nowadays most end-user devices come with multiple interfaces, *e.g.*, LTE and Wi-Fi. The bandwidth of the links connecting the clients to the ISP routers is based on the values

reported in the Netflix ISP Speed Index [62]. In detail, the bandwidth of each link is randomly selected from a normal distribution, with a mean close to the average ISP speed reported in the Netflix ISP Speed Index, and standard deviation 1.5. It is worth noting that the standard deviation reported in the Netflix ISP Speed Index tends to be much lower than 1.5, but we choose this value in order to allow more client diversity. The distribution of the total bandwidth of the clients used in the simulations, considering the two interfaces on the clients, is shown in Fig. 4.5.

In average, the used topologies have more than 110 ISP routers and more than 230 clients, additionally to the source and the 10 IXP routers.

4.3.3 Evaluation Setup

We consider that the end-users are interested in a video v that is available in three different representations, $\mathcal{Q} = \{480p, 720p, 1080p\}$ with bitrates $\{1750kbps, 3000kbps, 5800kbps\}$, respectively, that are a subset of the ones used by Netflix in the past [2]. Each representation is divided into a set of 50 segments, each of a duration of 2 seconds. These segments are further divided into generations and Data packets, as presented in Section 4.2.1. When network coding is enabled, the coding operations are performed in a finite field of size 2^8.

To select the representation that better adapts to their condition, the clients use an adaptation logic that considers the throughput measured by the client and the number of video segments that are buffered. The adaptation logic used in our evaluation is based on the one used by the DASH reference client, *dash.js*. As it has been demonstrated [93] that the simple design of the *dash.js* adaptation logic performs better than other more sophisticated adaptation logics available in the literature.

To demonstrate the Interest aggregation capabilities of the NetCodNDN forwarder, we consider a scenario where all the clients start requesting the video segments within the first 100ms of the simulation, and then carry on until they receive all the video segments. It is worth noting that since each client has different access bandwidth, and its adaptation logic works independently from other clients, the requested representation and segment IDs may vary across the clients. This means that at the same moment, different clients may be requesting different segments and representations, and thus some clients may finish retrieving all the segments earlier than others.

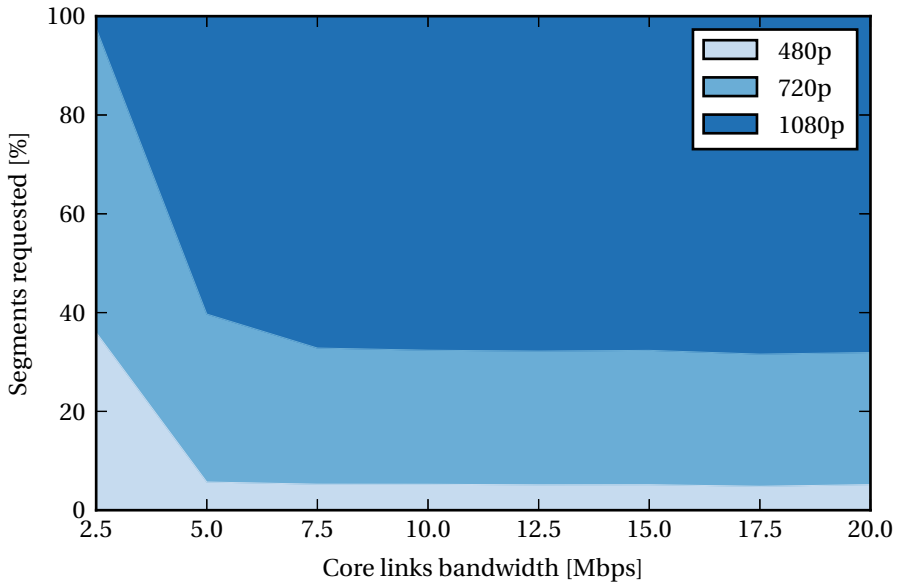

Figure 4.6: Representation of the segments received by the clients with respect to different core links bandwidth, for DAS-NetCodNDN.

4.3.4 Evaluation Results

In this section we evaluate and quantify the benefits that network coding brings to video streaming over NDN. First, we evaluate how efficiently the network bandwidth is used in DAS-NetCodNDN. Then, we evaluate the cache-hit rate at the routers and how network coding increases it. Finally, we evaluate the impact that the efficient use of the network bandwidth and the increased cache-hit rate has for the sources and for the clients.

Network Bandwidth Usage

We start by evaluating how the bandwidth of the links in the core network, *i.e.*, the links connecting the sources with the IXP routers, as well as the IXP routers with the ISP routers, affects the video quality received by the clients. Figs. 4.6 and 4.7 show the percentage of segments corresponding to each of the available video representations delivered for different values of the core links bandwidth, with DAS-NetCodNDN and DAS-NDN, respectively. As it can be seen in Fig. 4.6, the percentage of segments delivered at the highest representation available (*i.e.*, 1080p) whith DAS-NetCodNDN, stabilizes at around 70% of the total number of delivered segments for core links bandwidth higher than 7.5Mbps. However, as it can be seen in Fig. 4.7, with DAS-NDN

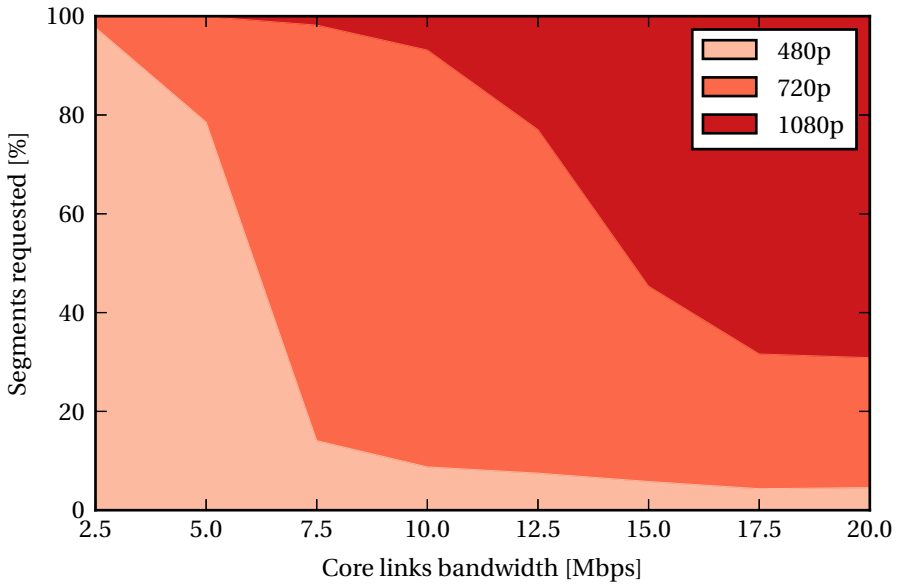

Figure 4.7: Representation of the segments received by the clients with respect to different core links bandwidth, for DAS-NDN.

the percentage of segments delivered at 1080p reaches the same stability only for core links bandwidth higher than 17.5Mbps. This behavior is observed as the use of network coding alleviates the competition between the clients for the core network resources. Thus, it is clear from these results that by using network coding, the network resources are more efficiently exploited. For the remainder of the evaluation, we consider that core links have a bandwidth of 10Mbps.

Cache-hit Rate

We now evaluate the cache-hit rate at the ISP and IXP layers. In Fig. 4.8, we can see that at the ISP layer, the cache-hit rate is constantly higher for DAS-NetCodNDN, as compared to DAS-NDN. After 60 seconds of streaming, the cache-hit rate for DAS-NDN is around 30%, while for DAS-NetCodNDN is around 50%, *i.e.*, about 20% higher. The reason for the lower cache-hit rate of DAS-NDN is that, since the clients are distributing the set of Interests that request a particular video segment over both of their faces, they need to coordinate the face over which each Interest is sent, so that they are aggregated at a router closer to the clients. However, due to the high granularity of the content (*i.e.*, each Data packet is unique and can satisfy only the Interest with the specific matching name), such coordination is not possible for each Data packet, as it requires centralized control. Further, it does not scale with the size

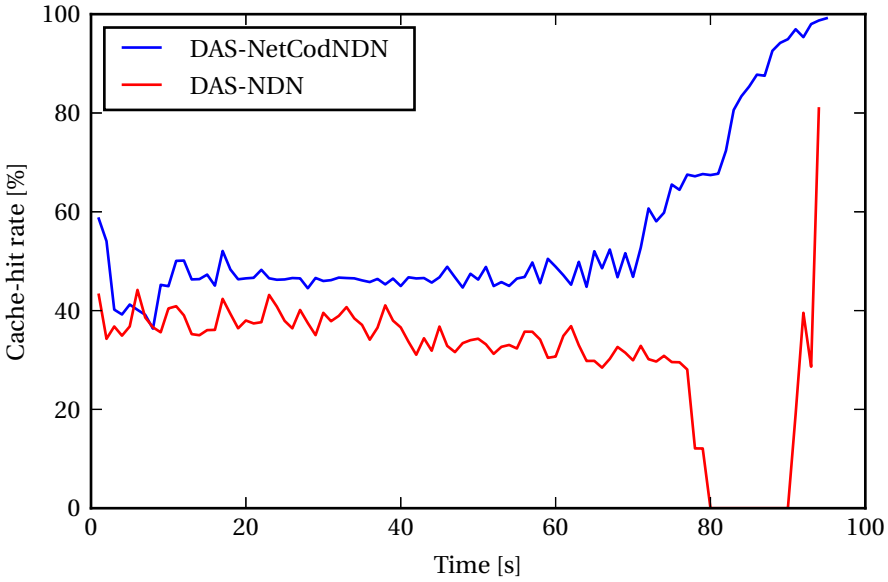

Figure 4.8: Cache-hit rate at the ISP layer.

of the network and the length of the video. On the contrary, the need for coordination is eliminated by introducing network coding, since clients do not send Interests for a particular Data packet, but for any network coded Data packet. Thus, in DAS-NetCodNDN an Interest can be satisfied with any innovative Data packet available in the CS of a node. These results verify our initial motivation for using network coding to enable a more efficient Interest aggregation and improve the use of the available bandwidth at the clients through efficient multipath communication.

At the IXP layer, both the traditional and the network coded architectures show a higher cache-hit rate than the one achieved at the ISP layer. After 60 seconds of streaming, the cache-hit rate for DAS-NDN is around 75%, while for DAS-NetCodNDN it is around 95%, as illustrated in Fig. 4.9. The cache-hit rate is high at the IXP layer because the 10 routers that belong to this layer are receiving Interests from more than 200 clients, meaning that the probability of aggregating Interests at this layer is higher than at the ISP layer. The increased cache-hit rate of the network coding architecture has two major performance consequences: *(i)* the number of Interests that reach the source is reduced and, *(ii)* the data bitrate seen by the client increases, since the Data packets are found closer to them.

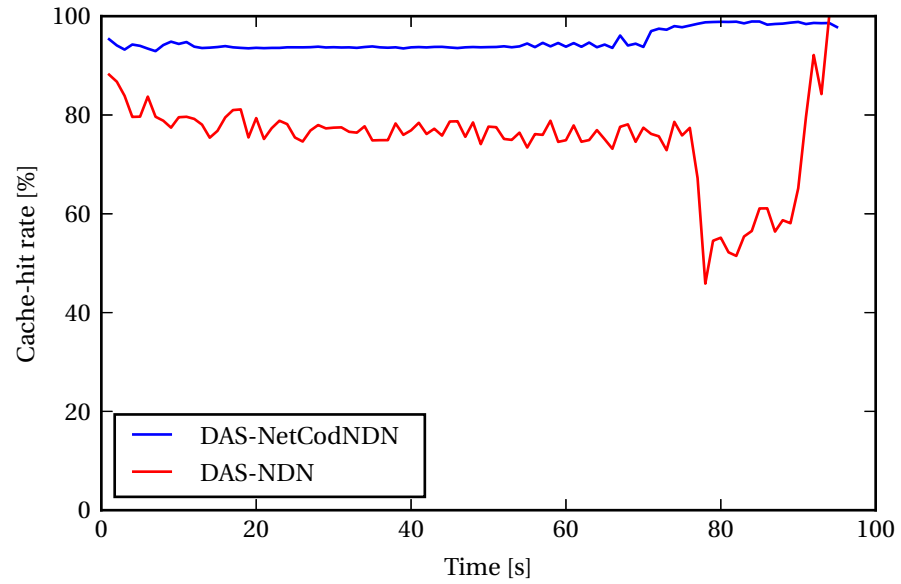

Figure 4.9: Cache-hit rate at the IXP layer.

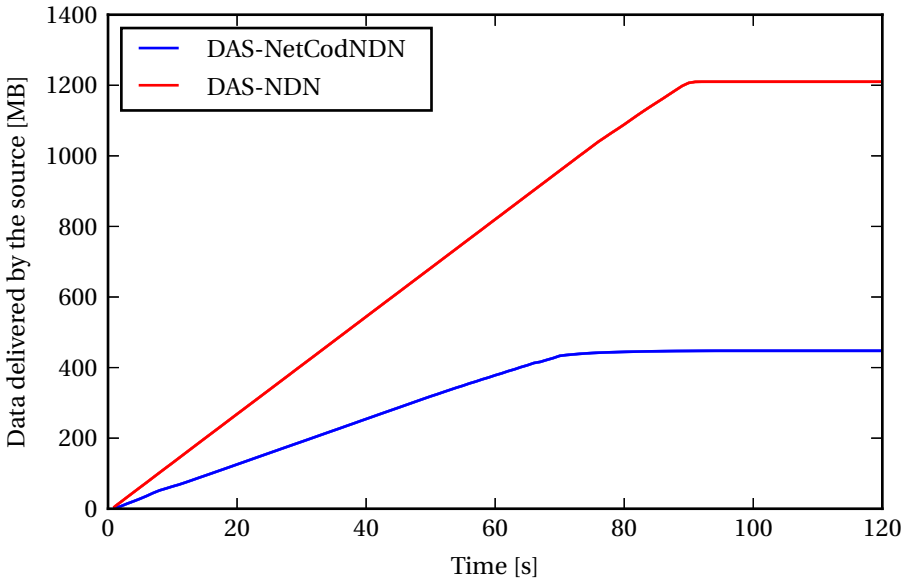

Figure 4.10: Total data delivered by the source.

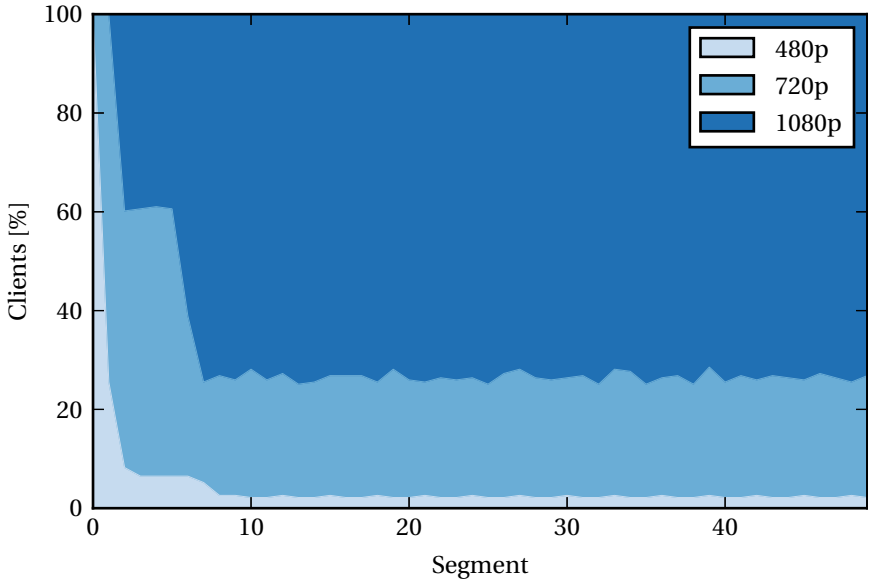

Figure 4.11: Representations requested by the clients with DAS-NetCodNDN.

Impact for the Sources

First, let us investigate the impact of using network coding for the source. In Fig. 4.10, we can see that there is a decrease in the total number of bytes that are provided by the source, from 1200MB for DAS-NDN to 450MB for DAS-NetCodNDN. This means that the use of network coding decreases the load on the source by more than 60%, *i.e.*, NDN imposes almost 3 times more load on the sources that DAS-NetCodNDN. This source load reduction translates into lower costs for the video content provider, that can reduce the number of servers at its data centers and reduce the amount of bandwidth needed.

Impact for the Clients

We now examine the benefits that network coding brings to the clients. The percentage of clients that decide to request a given video representation is shown in Figs. 4.11 and 4.12, for DAS-NetCodNDN and DAS-NDN, respectively. With DAS-NetCodNDN more than 70% of the clients are able to retrieve the video in the highest quality available (1080p), after a short adaptation period of around 8 segments. In contrast, with DAS-NDN less than 20% of the clients are able to receive the same quality, and only after a long adaptation period of around 45 segments. The reason for this is that since more Data packets are served from closer caches, the goodput measured by

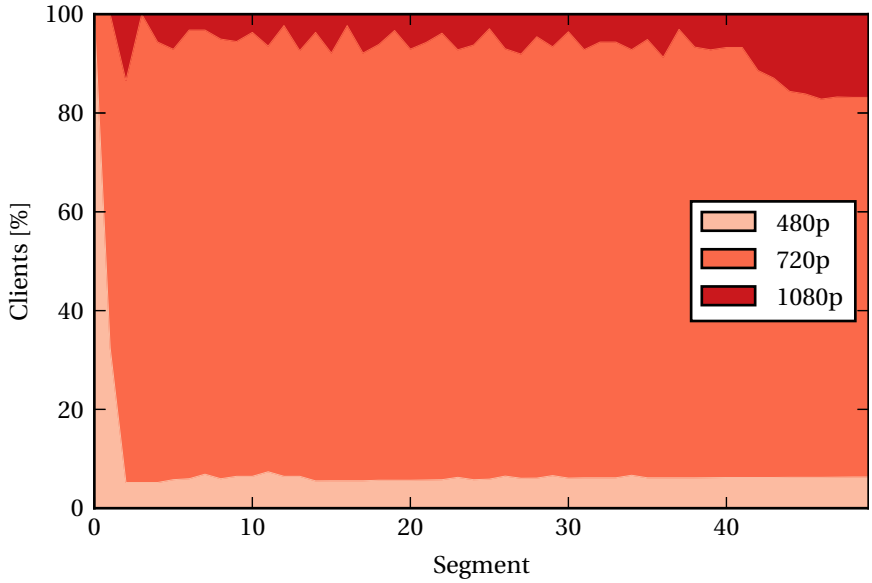

Figure 4.12: Representations requested by the clients with DAS-NDN.

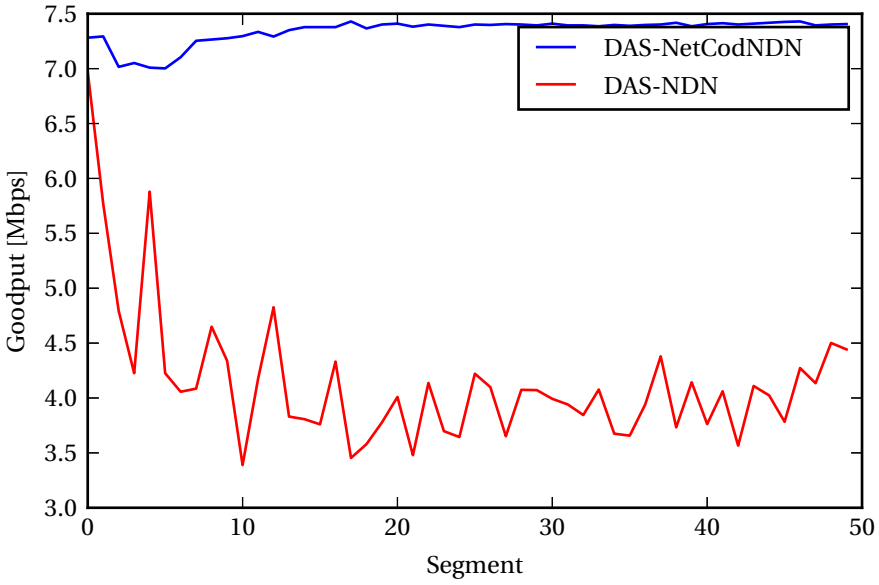

Figure 4.13: Average goodput measured by the clients.

the clients is higher with DAS-NetCodNDN, as illustrated in Fig. 4.13. Specifically, the goodput measured by the DAS-NetCodNDN clients is around 7.5Mbps, while it is approximately 4Mbps for DAS-NDN clients. This is inline with the average client bandwidth of 8Mbps, which after excluding the protocol overhead data, leaves a goodput of around 7.5Mbps. The increased goodput accelerates the DASH adaptation towards the highest representation, meaning that the clients are able to receive the best possible quality earlier.

It is worth mentioning that the DAS-NetCodNDN clients enjoy an increased goodput compared to DAS-NDN clients and hence improved quality, despite the fact that the implementation of our DAS-NetCodNDN client introduces a small delay between the requests of two consecutive generations. This delay is due to the fact that the DAS-NetCodNDN client needs to wait until a generation is decoded before requesting the next one, meaning that no Data packets are flowing to the client in the time interval between the arrivals of the last Data packet of generation g and the first Data packet of generation $g+1$. The DAS-NDN client does not suffer from this problem, as it does not consider generations. The goodput achieved by our architecture can be further improved by a more advanced client implementation, that would allow the clients to start requesting the next generation before decoding the current one. This has been successfully deployed in previous works [13, 90] for video streaming in host-centric networks deploying generation-based network coding.

4.4 Conclusions

In this Chapter we have presented DAS-NetCodNDN, a Dynamic Adaptive Video streaming architecture over NetCodNDN, based on DASH. Our architecture takes advantage of multi-path communication and uses network coding to eliminate the need for coordination between the network nodes. This improves the quality of the delivered video, reduces the resource utilization at the sources and improves the resiliency to Data packet erasures. We implemented our architecture by modifying the original NDN codebase, to enable network coding operations at the sources, routers, and clients. We evaluated our architecture using a network topology similar to the one used by video content providers. We have observed that network coding brings large performance gains in terms of the load on the source, as well as an increased cache-hit rate.

We can conclude that introducing network coding into NDN brings significant gains to video content providers, by reducing the traffic load on the servers and improving the use of network resources. This will, in turn, have an impact on the cost for the video content providers, leading to reduced cost for the video content consumers. Moreover, network coding also improves the speed at which the clients obtain the desired video representation.

The study presented in this Chapter considers that the routers have enough capacity on their content stores to cache all the Data packets that they receive. This allows focusing on the study of the benefits that network coding brings to NDN, without the need to consider the effects that particular caching policies have on this. However, this is not practical for real-life scenarios. Thus, in the next Chapter we propose a caching policy tailored for network coded data retrieval over NDN.

5

Caching Policy for Network Coding Enabled NDN

5.1 Introduction

Chapter 3 presented the NetCodNDN architecture, which enables efficient content retrieval in multi-source and multi-client scenarios for data intensive applications over NDN. Then, in Chapter 4 the NetCodNDN architecture was advanced to support adaptive video streaming and show the benefits that network coding brings to data intensive applications over NDN. However, these two Chapters considered scenarios in which the routers have unlimited storage capacity in their content store to cache all the Data packets that they receive, which is impractical in real-life networks.

When routers have limited caching storage, a caching policy is needed in order to decide which Data packets are placed into the cache (*placement*), as well as which data packets are evicted from the cache when the cache is full and a new Data packet should be cached (*eviction*). The goal of this Chapter is to develop a distributed caching policy that preserves the benefits that network coding brings to NDN for the

realistic scenario in which the content stores have limited capacity.

To this aim, this Chapter proposes *PopNetCod* [79], a popularity-based caching policy for the NetCodNDN architecture. PopNetCod is a caching policy in which routers distributedly estimate the popularity of the content objects based on the received Interests. Based on this information, each router decides the number of Data packets for each name prefix that it caches or evicts from its content store. The decision to cache Data packets is taken while processing the Interests. Since the first routers to process Interests in their path to the source are the edge routers, this helps to cache the most popular Data packets closer to the network edges, which reduces the data delivery delay [24, 26, 87]. To avoid that two independent routers cache the same Data packet, when a router decides to cache the Data packet that is expected as a reply to a received Interest, it informs the routers upstream in the path by setting a binary flag in the Interest. This increases the Data packet diversity in the caches. Then, whenever a router receives a Data packet with a name that the router previously decided to cache, it stores the Data packet in its content store and informs its downstream routers that it has cached this Data packet. This helps to avoid caching duplicated Data packets in the path, since the decision to cache a number of Data packets is taken for name prefixes that include multiple network coded Data packets. If the cache of a router is full and a Data packet should be cached, the router decides which Data packet should be evicted from its content store based on the popularity information.

The proposed caching policy has been implemented on top of the updated version of NetCodNDN presented in Chapter 4. We evaluate the performance of PopNetCod in a Netflix-like video streaming scenario, designed using parameters available in the literature [2, 12, 62]. In comparison with a caching policy that uses the NDN's default placement policy, *i.e.*, Leave Copy Everywhere (LCE), and the widely popular Least Recently Used (LRU) eviction policy, PopNetCod is able to achieve a higher cache-hit rate, which preserves better the benefits of network coding reported in Chapter 4, *i.e.*, higher video quality at the clients and reduced load at the sources.

This Chapter is organized as follows. Section 5.2 introduces the problem of caching in network coding enabled NDN for data intensive applications. Then, Section 5.3 presents our caching policy, PopNetCod. A practical implementation of the PopNetCod caching policy is described in Section 5.4. Section 5.5 presents the evaluation of the PopNetCod caching policy.

5.2 Caching in Network Coding Enabled NDN

Whenever a NetCodNDN router r receives an Interest $\hat{i}_{n,g}$ over face f, it can either *(i)* reply with a Data packet $\hat{p}_{n,g}$, if it can generate a network coded Data packet that has high probability of being innovative to its neighboring node connected over face f, or, otherwise, *(ii)* it forwards the Interest $\hat{i}_{n,g}$ upstream.

Recall from Chapter that $\xi^f_{n,g}$ denotes the number of network coded Data packets that the router can generate with the content of its CS and that have high probability of being innovative to its neighboring node connected over face f. Then, if at time t the router r receives the Interest $\hat{i}_{n,g}$, a cache-hit is defined as:

$$h^f_{n,g}(t) = \begin{cases} 1, & \text{if } \xi^f_{n,g} > 0 \\ 0, & \text{otherwise.} \end{cases} \tag{5.1}$$

Let us now assume that during a time period $[t, t+T]$ the router r receives a set of Interests $\mathscr{I}(t,T)$. The cache-hit rate during this time period is defined as follows:

$$H(t,T) = \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{5.2}$$

The overall cache-hit rate seen by router r at time t can be computed as follows:

$$H(t) = \lim_{T\to\infty} H(t,T) = \lim_{T\to\infty} \frac{1}{T} \sum_{t'=t}^{t+T} h^f_{n,g}(t'). \tag{5.3}$$

To make optimal use of the limited CS capacity while maintaining the benefits that network coding brings to NDN, the objective of each router is to maximize the number of Interests that it can satisfy with the Data packets available in its CS, *i.e.*, maximize its overall cache-hit rate. Achieving a high cache-hit rate at the routers is beneficial for both clients and sources. For the sources, an increased cache-hit reduces their processing load and bandwidth needs, since the number of Interests that they receive is reduced. For the clients, the delivery delay is reduced, since the Interests are satisfied with Data packets cached at routers closer to them.

It is clear from (5.1), (5.2), and (5.3) that in order to maximize the overall cache-hit rate, routers should maintain the value of $\xi^f_{n,g}$ high enough so that most of the Interests

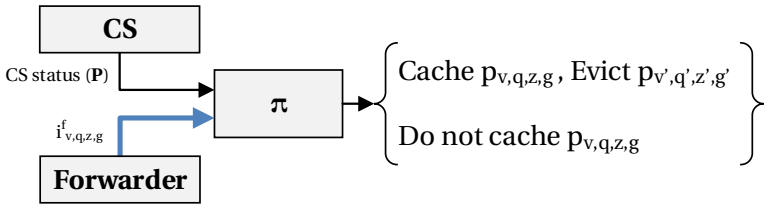

Figure 5.1: Overview of the caching policy.

received can be satisfied with the Data packets on their CS. However, since in this Chapter we consider that the routers' CS have limited capacity, it is unfeasible for a router to cache all the Data packets that it receives, as it has been considered in Chapters 3 and 4 of this thesis, and in the literature [59, 78, 80]. Optimal solutions to this issue have been proposed [49, 97] and consider a central controller that knows the network topology and is aware of all the Interests received by the routers. However, these solutions do not scale well with the size of the network, since they require a high number of signaling messages and a powerful enough controller. Hence, in this Chapter we consider that each router decides in an online manner if a Data packet should be cached or not, and which Data packets should be evicted from the CS when it is full. This is achieved by using a distributed caching policy π that, based on the content store status (*i.e.*, the Data packets that are currently cached and the available space) and the context (*e.g.*, previous requests, time of the day, *etc.*), decides which Data packets should be cached and which ones should be evicted from the content store in order to maximize the overall cache-hit rate $H(t)$ of each router, as shown in Fig. 5.1. Thus, the goal of the router can be described as:

$$\max_{\pi} H(t). \tag{5.4}$$

The optimal caching policy π should predict which Interests will be received in the future, so that the router caches the Data packets that will be useful to satisfy those Interests.

5.3 The PopNetCod Caching Policy

In this section, we present our popularity-based caching policy for NetCodNDN, called PopNetCod. To increase the overall cache-hit rate, the PopNetCod caching policy exploits real-time data popularity measurements to determine the amount

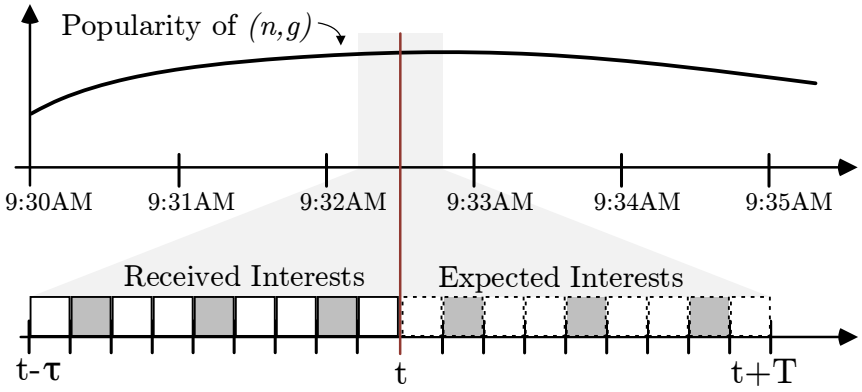

Figure 5.2: Popularity prediction for the name prefix (n, g).

of Data packets that each router should cache for each name prefix. In order to determine which Data packets to cache and evict, such that the overall cache-hit rate is maximized, the PopNetCod caching policy performs the following steps. First, it measures the popularity of the different name prefixes contained in the Interests that pass through it. Then, it uses this popularity measurements to predict the Interests that it will receive. Finally, using this prediction, it determines in an online manner the Data packets that should be cached and the ones that should be evicted from the CS.

5.3.1 Popularity Prediction

The popularity prediction in PopNetCod is based on the fact that the rate $\lambda^f_{n,g}(t)$ at which Interests for a particular content object arrive at a router r over face f at time t tends to vary smoothly, as shown in Fig. 5.2. Thus, the router r can predict the rate of the Interests that it will receive in the near future by observing the Interests that it has recently received. Let us denote $\mathscr{I}^f_{n,g}(\tau, t)$ as the set of Interests for the name prefix (n, g) that the router r has received over face f during the past period $[t-\tau, t]$, where t is the current time and τ is the observation period. Let us also denote $\mathscr{I}^f(\tau, t)$ as the total set of Interests for all the name prefixes received over face f during the period $[t-\tau, t]$. Using the sets $\mathscr{I}^f_{n,g}(\tau, t)$ and $\mathscr{I}^f(\tau, t)$, the router r can compute the average Interest rate for the name prefix (n, g) over face f as follows:

$$\lambda^f_{n,g}(\tau, t) = \frac{|\mathscr{I}^f_{n,g}(\tau, t)|}{|\mathscr{I}^f(\tau, t)|}, \tag{5.5}$$

Note that since the average Interest rate does not vary abruptly, the average Interest rate $\lambda^f_{n,g}(\tau, t)$ of the recent period $[t-\tau, t]$ will be very close to that expected in the near future, *i.e.*, in the period $[t, t+T]$ where T is the length of the prediction period. Thus, $\lambda^f_{n,g}(\tau, t) = \lambda^f_{n,g}(t, T)$, which hereafter we denote as $\lambda^f_{n,g}(t)$. The PopNetCod caching policy uses $\lambda^f_{n,g}(t)$ to predict the number of Interests with name prefix (n, g) that will be received over face f in the near future, and hence, to allocate more storage space in the CS to Data packets with higher probability of cache-hit.

In order to prepare the CS for the Interests that the router may receive, the PopNetCod caching policy maps the received Interest rate to the capacity of the CS, such that name prefixes with high Interest rate are allocated more space in the content store. The number of network coded Data packets with name prefix (n, g) that the router should cache in its CS at time t to satisfy the Interests expected over face f is denoted as $M^f_{n,g}(t)$ and computed as:

$$M^f_{n,g}(t) = \begin{cases} \lambda^f_{n,g}(t) \cdot M, & \text{if } \lambda^f_{n,g}(t) \cdot M < |\hat{\mathscr{P}}_{n,g}| \\ |\hat{\mathscr{P}}_{n,g}|, & \text{otherwise,} \end{cases} \tag{5.6}$$

where M is the capacity of the CS.

5.3.2 PopNetCod Placement

In the PopNetCod caching policy, the placement decision is taken following the reception of an Interest. Whenever a router decides to cache the Data packet that is expected as reply to the received Interest, it sets a flag on the Interest signaling upstream routers about its decision. In this way, the upstream nodes do not consider this Interest for caching. Since the edge routers (*i.e.*, the routers that are directly connected to the clients) are the first ones that have the possibility to decide whether they will cache a Data packet, the PopNetCod caching policy naturally enables edge caching. This is inline with recent works [24, 26, 87] arguing that most of the gains from caching in NDN networks come from the edge caches, and thus, it is natural to cache the most popular content at the edge routers.

Whenever a router receives an Interest $\hat{i}_{n,g}$ over face f_t at time t, the PopNetCod caching policy follows the next steps to decide if the Data packet $\hat{p}_{n,g}$ should be cached. First, it uses the popularity prediction to compute $M^f_{n,g}(t)$, *i.e.*, the total

number of Data packets that it aims to cache for the name prefix (n, g), as defined in (5.6). Then, it computes the number of Data packets that it should cache in order to satisfy the expected Interests as:

$$\delta^f_{n,g}(t) = M^f_{n,g}(t) - \xi^f_{n,g} \forall f \in \mathscr{F}, \tag{5.7}$$

where $\xi^f_{n,g}$ is the number of network coded Data packet that the router can generate and that have high probability of being innovative to its neighboring node connected over face f.

Finally, the caching policy decides to cache the Data packet $\hat{p}_{n,g}$ that is expected as reply to the received Interest if the average number of Data packets needed by all the faces is greater than 0. However, it should be noted that the Data packet $\hat{p}_{n,g}$ will not be useful to the node connected over the downstream face f_t over which the Interest arrived. This is because when the Data packet $\hat{p}_{n,g}$ arrives at the router, it will be sent to face f_t in order to satisfy the received Interest. Then, replying with the same Data packet to a subsequent Interest received over the same face f_t will not add any innovative information, *i.e.*, the Data packet will be considered duplicated. Instead, the expected Data packet $\hat{p}_{n,g}$ is potentially useful for all the nodes connected over all the other downstream faces of the router. For this reason, the average number of Data packets needed is taken only over the downstream faces different to the one over which the Interest arrived. It is computed as:

$$\Delta^+_{n,g}(t) = \frac{1}{|\mathscr{F}^r| - 1} \sum_{\substack{f \in \mathscr{F} \\ f \neq f_t}} \delta^f_{n,g}(t) > 0. \tag{5.8}$$

5.3.3 PopNetCod Eviction

When a router receives a network coded Data packet that it has previously decided to cache, but its CS is full, it first should evict at least one Data packet from the CS before caching the recently arrived one.

The steps followed by the PopNetCod caching policy to decide how many Data packets with name prefix (n, g) can be evicted from the router's CS are the following. Similarly to the placement case, first, the caching policy uses the popularity prediction to compute $M^f_{n,g}(t)$, *i.e.*, the number of Data packets that it aims to cache for the name

prefix (n,g). Then, it computes the number of Data packets that it can evict from its CS and still satisfy the expected Interests as:

$$\tilde{\delta}^{f}_{n,g}(t) = \texttt{rank}(\hat{\mathbf{P}}^{r}_{n,g}) - M^{f}_{n,g}(t) \forall f \in \mathscr{F}. \tag{5.9}$$

Finally, the number of Data packets the router can evict from a particular name prefix (n,g) is computed as the minimum number of Data packets that it can evict over all the faces:

$$\Delta^{-}_{n,g}(t) = \min_{f \in \mathscr{F}} \tilde{\delta}^{f}_{n,g}(t). \tag{5.10}$$

5.4 Practical Implementation of PopNetCod

In this section, we describe a practical implementation of the PopNetCod caching policy for the NetCodNDN architecture described in Chapters 3 and 4. In order to enable the use of caching policies in the NetCodNDN architecture, we extend its design by adding a new module called Content Store Manager (CSM). The CSM manages the content store by enforcing a determined caching policy. In the following sections, we describe the functioning of routers which CSM is configured with the PopNetCod caching policy. First, we describe the signaling between routers. Even if each router makes the caching decision in a completely distributed manner, Interests and Data packets carry a binary flag that prevents routers of the same path to cache duplicate Data packets. Next, we present the Interest processing algorithm, where placement decisions are made. Finally, we describe the Data packet processing algorithm for placement enforcement, eviction decision, and eviction enforcement.

5.4.1 Signaling Between Routers

As described in Section 5.3, the PopNetCod caching policy is distributed and requires very little signaling between the different routers. The only signaling that exists between the PopNetCod caching policy is a binary flag added to the Interest and Data packets that is used to inform neighbor routers that an expected Data packet will be cached or that a received Data packet has been cached. Distributed caching policy decisions helps to keep the complexity of the system low and to make our system scalable to a large number of routers.

Each Interest $\hat{i}_{n,g}$ carries a flag `CachingDown`, which is set to 1 by a CSM when it decides to cache the Data packet $\hat{p}_{n,g}$ that is expected to come as reply to the Interest. This flag informs the upstream routers that another router downstream has already decided to cache the Data packet that is expected to come as reply to this Interest. The routers receiving an Interest with the `CachingDown` flag set to 1 do not consider to cache the Data packet that is expected to come as reply to this Interest, therefore reducing the amount of duplicated Data packets in the path and the processing load in the nodes.

Since Interests for network coded data do not request particular Data packets, but rather any network coded Data packet with the requested name prefix, the routers need a way to know that a Data packet has been already cached by another router, so that they avoid caching duplicated Data packets. For this reason, each Data packet $\hat{p}_{n,g}$ has a flag `CachedUp`, which is set to 1 by a CSM when it caches this Data packet in its CS. This flag informs the downstream routers that another router has already cached this Data packet. A router receiving a network coded Data packet with the `CachedUp` flag set to 1 does not consider it for caching. Instead, it will wait for another Data packet with the same name prefix that has not ben cached upstream. This ensures that a Data packet is cached by only one router on its way to the client.

5.4.2 Status Information at Routers

Each CSM configured with the PopNetCod caching policy should store information that assist to identify the Data packets that should be cached or evicted. In particular, the CSM needs to keep the *Recently received Interests* information to compute the popularity prediction. Moreover, since the placement decision takes place when the Interest is received, the CSM needs to remember the *Names to be cached*, such that the selected Data packets are cached when they arrive. Finally, since the popularity information can vary over time, the CSM should keep a list with the *Names to consider for eviction*, which is used when they decide about eviction. Below, we describe the data structures used to store this information.

- *Recently received Interests* — The CSM maintains a list $\mathbf{L}^f$ for each face f of the router, where it stores the names of the Interests $\mathscr{I}^f(\tau, t)$ received over face f during the period $[\tau, t]$. The parameter τ controls how much of the recent past the router observes. Together with the name prefix, each element in $\mathbf{L}^f$ also

keeps the time t_i at which the Interest was received, such that it can be removed from the list at time $t_i + \tau$.

- *Names to be cached* — The CSM maintains a table **A**, where it stores the name prefixes (*i.e.*, the content object name appended with the generation ID) of the Data packets that should be cached, together with the number of Data packets that should be cached for each name prefix. The table **A** is implemented as a hash table, in order to provide fast insert and search operations. When the router receives an Interest $\hat{i}_{n,g}$ and the PopNetCod caching policy decides that the network coded Data packet that is expected as reply should be cached, the CSM adds its name prefix (n,g) to the table **A**, or increases its counter if the name prefix (n,g) already exists on the table. Then, whenever a network coded Data packet arrives, the CSM looks for the name prefix of the Data packet in the table **A**. If it finds a match, it caches the Data packet, and reduces the counter value by 1.

- *Names to consider for eviction* — The CSM also maintains a queue **E**, where it stores the name prefixes (*i.e.*, the content object name appended with the generation ID) of the CS entries that can be considered for Data packet eviction. When a name prefix (n,g) is removed from the list $\mathbf{L}^f$, the popularity of this name prefix decreases, *i.e.*, it becomes a good candidate to consider for eviction. Thus, each time a name prefix is removed from $\mathbf{L}^f$, it is added to **E**. Then, whenever the CSM needs to evict a Data packet from the CS to cache a new one, it chooses a name prefix from **E**, computes the number of Data packets that can be evicted from the CS entry, and then proceeds to evict the computed number of Data packets.

5.4.3 Interest Processing

As depicted in Fig. 5.3, when a CSM configured with the PopNetCod caching policy receives an Interest $\hat{i}_{n,g}$ from downstream, it *(i)* determines if the Interest can be replied from the CS. Then, if the CSM could not reply to the Interest with the content of its CS, it *(ii)* updates the popularity information, and, *(iii)* determines if the Data packet that is expected as reply to this Interest should be cached. The CSM should provide the NetCodNDN forwarder with either a Data packet that should be sent downstream as reply to the Interest, or an Interest that should be forwarded upstream. Below we describe the details of this procedure, which is summarized in Algorithm 5.

Algorithm 5 Interest processing at the CSM

Require: $\hat{i}_{n,g}$, f

$t \leftarrow$ current time
if Flag `CachingDown` in $\hat{i}_{n,g}$ is set to 1 **then**
  if $\xi^f_{n,g} > 0$ **then** ($\hat{i}_{n,g}$ *can be satisfied from the CS*)
    Generate a Data packet $\hat{p}_{n,g}$ from the CS
    Return $\hat{p}_{n,g}$
  else
    Return $\hat{i}_{n,g}$
  end if
else
  Add (n,g) to $\mathbf{L}^f$
  if $\xi^f_{n,g} > 0$ **then** ($\hat{i}_{n,g}$ *can be satisfied from the CS*)
    Generate a Data packet $\hat{p}_{n,g}$ from the CS
    Return $\hat{p}_{n,g}$
  else if $\hat{i}_{n,g}$ will be aggregated by the PIT **then**
    Return $\hat{i}_{n,g}$
  else
    Update **L**. (*Algorithm 6*)
    if $\Delta^+_{n,g}(t) > 0$ **then** ($\hat{p}_{n,g}$ *should be cached*)
      Insert (n,g) into **A**
      Set the flag `CachingDown` of $\hat{i}_{n,g}$ to 1
      Return $\hat{i}_{n,g}$
    else
      Return $\hat{i}_{n,g}$
    end if
  end if
end if

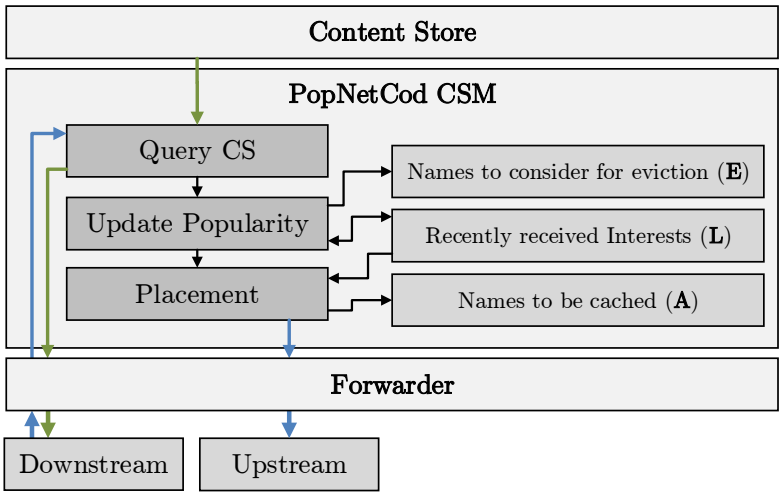

Figure 5.3: Access to the CS and the Status Information during the Interest processing in a CSM configured with the PopNetCod caching policy.

After receiving an Interest $\hat{i}_{n,g}$, the CSM first checks the flag `CachingDown` to see if any previous node downstream in the path has decided to cache the Data packet that is expected as reply to this Interest (lines 2 to 8). If the flag `CachingDown` is set to 1, then the CSM only checks its CS to determine if the Interest can be satisfied from the CS. If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder, which sends it over face f. If the Interest can not be satisfied from the CS, the CSM provides the same Interest to the NetCodNDN forwarder, which will forward it upstream.

Algorithm 6 Update **L**

1: **for all** $f \in \mathscr{F}^r$ **do**
2: **for all** expired entries (n_l, g_l) in $\mathbf{L}^f$ **do**
3: Remove (n_l, g_l) from $\mathbf{L}^f$
4: Add (n_l, g_l) to **E**
5: **end for**
6: **end for**

If the flag `CachingDown` is set to 0, the CSM first inserts the name (n, g) of the Interest into the list $\mathbf{L}^f$ (line 10). Then, the CSM checks if it can satisfy the Interest with the content of the CS (lines 11 to 13). If this is possible, *i.e.*, if $\xi^f_{n,g}$ is greater than 0, it generates a network coded Data packet from the CS and provides it to the NetCodNDN forwarder which sends it over face f. Otherwise, the node needs to forward the

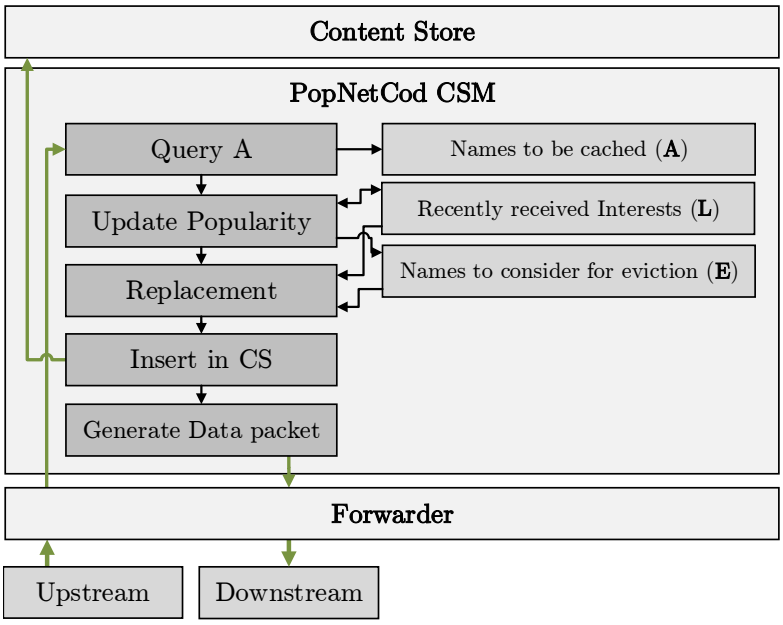

Figure 5.4: Access to the CS and the Status Information during the Data packet processing in a CSM configured with the PopNetCod caching policy.

Interest to its neighbor nodes. If the router will not send the Interest upstream, but will aggregate it in the PIT with a previously received Interest, the CSM does not need to do anything else and provides the Interest to the NetCodNDN forwarder, which will aggregate it (line 15).

If the Interest will not be aggregated, then the CSM determines if it will cache the Data packet with name prefix (n,g) that is expected as reply to this Interest, by computing $\Delta^{-}_{n,g}(t)$ using Eq. (5.8). In order to obtain an accurate value of $\Delta^{-}_{n,g}(t)$, the CSM first updates the popularity information, removing all the expired elements from $\mathbf{L}^{f}$ and adding their name prefix to the list $\mathbf{E}$ of name prefixes to be considered for eviction (line 17). This procedure is summarized in Algorithm 6. Then, the CSM computes the value of $\Delta^{+}_{n,g}(t)$. If $\Delta^{+}_{n,g}(t)$ is greater than 0, it means that the Data packet should be cached. In this case, the CSM inserts the name prefix (n,g) into the table $\mathbf{A}$, sets the flag `CachingDown` on the Interest $\hat{i}_{n,g}$ to 1 and, finally, provides the modified Interest to the NetCodNDN forwarder, which will send it upstream (lines 18 to 21). If the value of $\Delta^{+}_{n,g}(t)$ is not greater than 0, then the CSM provides the same Interest to the NetCodNDN forwarder, which will forward it upstream (line 23).

Algorithm 7 Data packet processing at the CSM

Require: $\hat{p}_{n,g}$

if Flag `CachingUp` in $\hat{p}_{n,g}$ is set to 1 **then**
  Return $\hat{p}_{n,g}$
else if $(n,g) \notin \mathbf{A}$ **then**
  Return $\hat{p}_{n,g}$
else
  Update **A**
  if $|\mathscr{P}^r| == M$ **then** (*The CS is full*)
    Update **L** (*Algorithm 6*)
    while $|\mathscr{P}^r| == M$ **do**
      Select an element (n_e, g_e) from **E**
      if $\Delta^-_{n_e,g_e}(t) > 0$ **then**
        Evict $\Delta^-_{n_e,g_e}(t)$ Data packets with name prefix (n_e, g_e) from the CS
      end if
    end while
  end if
  Insert $\hat{p}_{n,g}$ into the CS
  Generate a Data packet $\hat{p}^*_{n,g}$ from the CS
  Set the flag `CachingDown` of $\hat{p}^*_{n,g}$ to 1
  Return $\hat{p}^*_{n,g}$
end if

5.4.4 Data Packet Processing

As depicted in Fig. 5.4, when a CSM configured with the PopNetCod caching policy receives a network coded Data packet $\hat{p}_{n,g}$ from upstream, it *(i)* determines if the Data packet should be cached in the CS, by consulting **A**. If the Data packet should be cached, the CSM ensures that there is enough free space in the CS, *(ii)* updating the popularity information and *(iii)* executing the cache replacement procedure if needed. Finally, the CSM *(iv)* inserts the received Data packet into the CS, and *(v)* generates a new network coded Data packet that should be forwarded downstream. This procedure is detailed below and summarized in Algorithm 7.

After receiving a Data packet $\hat{p}_{n,g}$, the CSM first checks the flag `CachedUp` to determine if any router upstream has already cached this Data packet. If the flag `CachedUp` has been set to 1, then, the CSM understands that another router upstream has already cached this Data packet. In this case, the CSM returns the Data packet to the NetCodNDN forwarder, which will reply to any matching pending Interest (line 1).

When the flag `CachedUp` is set to 0, then the CSM first verifies if any entry in **A** matches the name prefix (n,g). If there is no matching entry, the CSM returns the Data packet to the NetCodNDN forwarder (line 3). If there is a match, the Data packet should be cached, and **A** is updated by decreasing the counter of the matching entry by one (line 6). However, if the CS is full, the CSM first needs to free some space in the CS (lines 7 to 15). To evict Data packets, the CSM goes through the list **E**, each time selecting a name prefix (n_e, g_e) and computing the number of Data packets that can be evicted for the name prefix using Eq. (5.10). If this number is greater than 0, then the CSM evicts the corresponding number of Data packets from the CS and interrupts the scan of the list. Note that, since the cached Data packets are network coded, the CSM does not need to decide which particular Data packets from the CS entry $\hat{\mathscr{P}}_{n,g}$ it should evict from the CS, but it can rather select random network coded Data packets from the CS entry and evict them. After evicting at least one Data packet, the CSM caches the received Data packet $\hat{p}_{n,g}$. Then, the router generates a new Data packet $\hat{p}^*_{n,g}$ by applying network coding to the cached Data packets with name prefix (n,g). Since the new Data packet $\hat{p}^*_{n,g}$ contains the cached Data packet $\hat{p}_{n,g}$, the router sets the flag `CachedUp` of $\hat{p}^*_{n,g}$ to 1. Finally, the router provides the Data packet $\hat{p}^*_{n,g}$ to the NetCodNDN forwarder, which will use it to reply to any pending Interest with name prefix (n,g).

5.5 Evaluation

In this section, we evaluate the performance of the PopNetCod caching policy in an adaptive video streaming architecture based on NetCodNDN. First, we describe the evaluation setup. Then, we present the caching policies with which we compare the PopNetCod caching policy. Finally, we show the performance evaluation results.

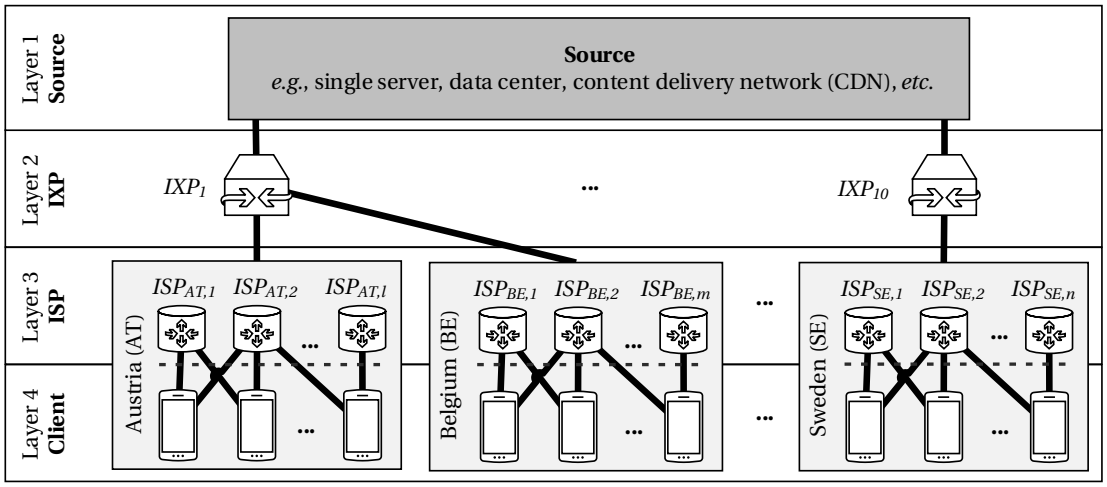

Figure 5.5: Layered topology used in the evaluation of PopNetCod.

5.5.1 Evaluation Setup

We consider a layered topology similar to the one used for the evaluations of the DAS-NetCodNDN architecture in Chapter 4, as the one depicted in Fig. 5.5. It consists of 1 source, 123 clients, and 45 routers connecting the clients and the sources. The routers are arranged in a two layer topology, with 10 routers directly connected to the source and 35 edge routers directly connected to the clients. The links connecting the routers between them and the links connecting the routers to the source have a bandwidth of $20Mbps$. The bandwidth of the links connecting the clients to the routers follow a normal distribution, with mean $4Mbps$ and standard deviation 1.5. These values are chosen based on the Netflix ISP Speed Index [62]. Each client is connected with two routers, considering that nowadays most end-user devices have multiple interfaces, *e.g.*, LTE, Wi-Fi, Bluetooth.

For the evaluation, we consider that the source offers 5 videos for streaming, each one composed of 50 video segments with a duration of 2 seconds each, *i.e.*, in total, each video has a duration of 100 seconds. The video segments are available in three different representations, $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$ with bitrates $\{1750kbps,\ 3000kbps,\ 5800kbps\}$, respectively. These values for the representations and bitrates are chosen according to the values that had been used by Netflix [2]. The content objects (*i.e.*, the video segments in our evaluation scenario) are divided into Data packets and generations, in order to implement network coding. In particular, for the representations $\mathcal{Q} = \{480p,\ 720p,\ 1080p\}$, each video segment is divided into $\{359, 615, 1188\}$ Data packets of 1250 bytes each, and $\{4, 7, 12\}$ generations, respectively.

Thus, in total, the source stores 540500 Data packets. All the routers are equipped with content stores able to cache the same number of Data packets, which in this evaluation is between 5000 to 12500 Data packets, *i.e.*, between 0.9% and 2.3% of the total Data packets available at the source.

The clients randomly choose a video to request and start the adaptive video retrieval process at a random time during the first 5 seconds of simulation. The network coding operations are performed in a finite field of size 2^8. The clients use the *dash.js* adaptation logic [93] to choose the representation that better adapts to the current conditions, *i.e.*, the measured goodput and the number of buffered video segments.

5.5.2 Benchmarks

We compare the performance of our caching algorithm with the following benchmarks:

- *LCE-NoLimit* — The placement policy is Leave Copy Everywhere (LCE). We consider that the CSs of the routers have enough space to store all the videos. This setting is similar to the one used in Chapters 3 and 4, which is used here to determine how much the different caching policies help to maintain the benefits of network coding in the video streaming system.
- *LCE+LRU* — The placement policy is NDN's default LCE, while the eviction policy is the widely used Least Recently Used (LRU), which evicts Data packets with the least recently requested name.
- *NoCache* — In this setting, the routers do not have a CS, *i.e.*, all the Data packets should be retrieved from the source.

5.5.3 Evaluation Results

Cache-hit Rate

We first evaluate the average cache-hit rate at the routers. In Fig. 5.6, we can see that by using the PopNetCod caching policy, the routers achieve a higher cache-hit rate than with LCE-LRU. This is because with PopNetCod the number of Data packets cached for a certain name prefix increases smoothly, according to the popularity. In comparison,

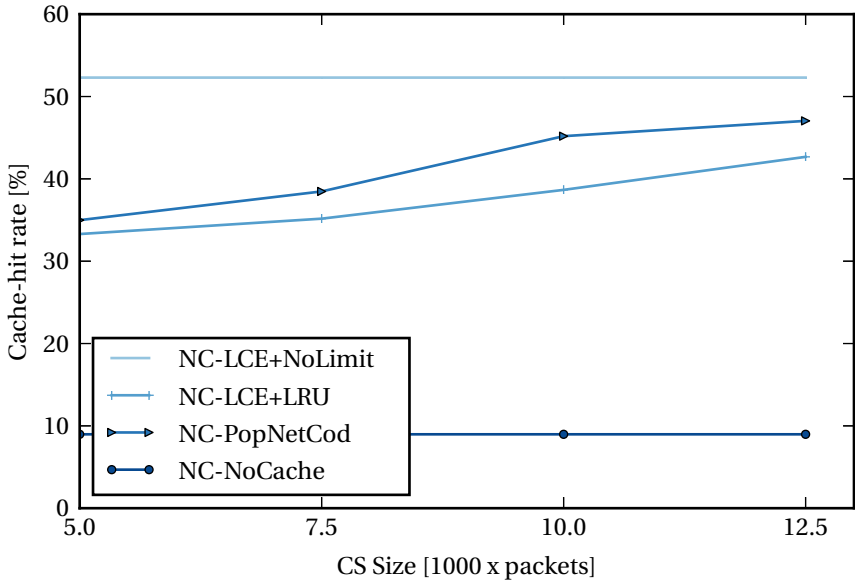

Figure 5.6: Average cache-hit rate in the routers.

with LCE+LRU all the Data packets received by the router are cached, and the least recently used are evicted from the CSs when the capacity is exceeded. Thus, if a router receives Data packets that are requested by a single client, the router still caches them, wasting storage capacity that could be used to cache more popular Data packets that are requested by multiple clients. We can also see that the LCE+NoLimit caching policy defines an upper bound to the cache-hit rate at the routers, since caching all the Data packets with unlimited CS capacity represents the best caching scenario. On the contrary, the NoCache case, where the routers do not have CS capacity, defines a lower bound to the cache-hit rate. Note that in our evaluation the NoCache policy has a non-zero cache-hit rate because our measurement of cache-hit rate also includes Interest aggregations, which is what is being measured in this case. As it can be seen in Fig. 5.6, the PopNetCod caching policy helps to maintain the benefits that network coding brings to NDN closer to what has been measured in Chapter 4 with the LCE+NoLimit caching policy.

The increased cache-hit rate that the PopNetCod caching policy brings to the routers has two major consequences: *(i)* the goodput at the clients increases, which enables the adaptation logic to choose higher quality representations when bandwidth is sufficient, and *(ii)* the source receives less Interests, meaning that its processing and network load is reduced.

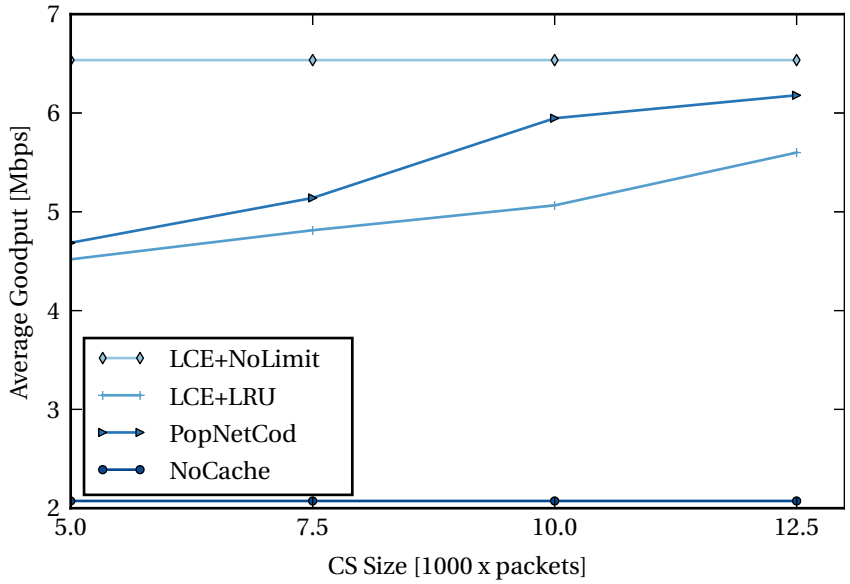

Figure 5.7: Average goodput perceived by the clients.

Impact for Clients

Let us first evaluate the impact that the increased cache-hit rate at the routers has for the clients. The percentage of video segments delivered to the clients for each of the available representations (*i.e.*, 480p, 720p, and 1080p) with the PopNetCod and LCE+LRU caching policies is shown in Figs. 5.8 and 5.9, respectively. We can see that, compared to the LCE+LRU policy, with the PopNetCod caching policy a higher percentage of video segments are delivered in the highest representation available, *i.e.*, 1080p. This happens because the Data packet retrieval time is reduced, since more Interests are being satisfied from the routers' content stores, which increases the goodput measured by the clients. The percentage of video segments delivered to the clients in each of the available representations with the upper bound LCE+NoLimit caching policy can be seen in Fig. 5.10.

Impact for Sources

Finally, we analyze the impact that the increased cache-hit rate in the routers has for the sources. Fig. 5.11 illustrates the load reduction on the source. This metric measures the percentage of Data packets received at the clients that have not been directly provided by the source. It is computed as $1 - N_{\mathscr{S}}^{sent}/N_{\mathscr{C}}^{rcvd}$, where $N_{\mathscr{S}}^{sent}$ denotes the total number of Data packets sent by the source, and $N_{\mathscr{C}}^{rcvd}$ denotes the

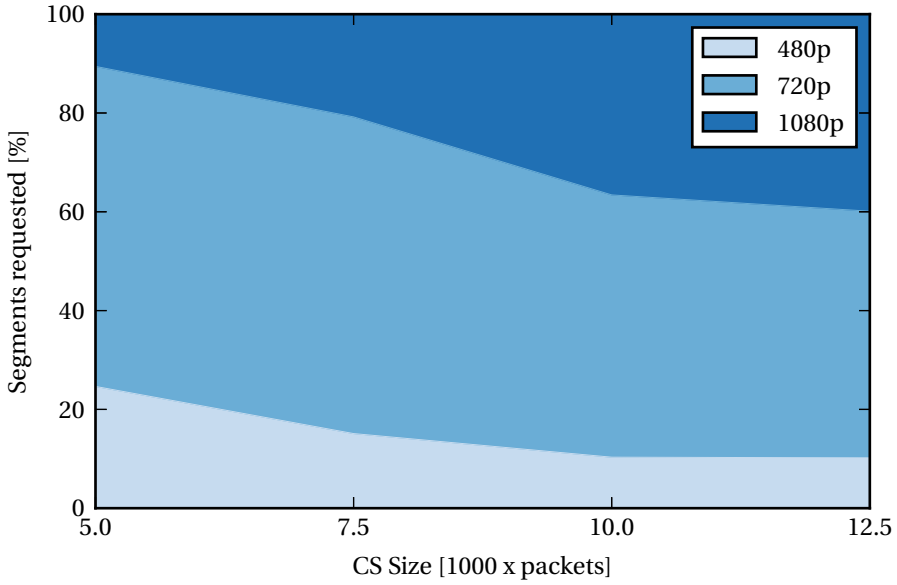

Figure 5.8: Percentage of video segments delivered in each of the available representations, with the PopNetCod caching policy.

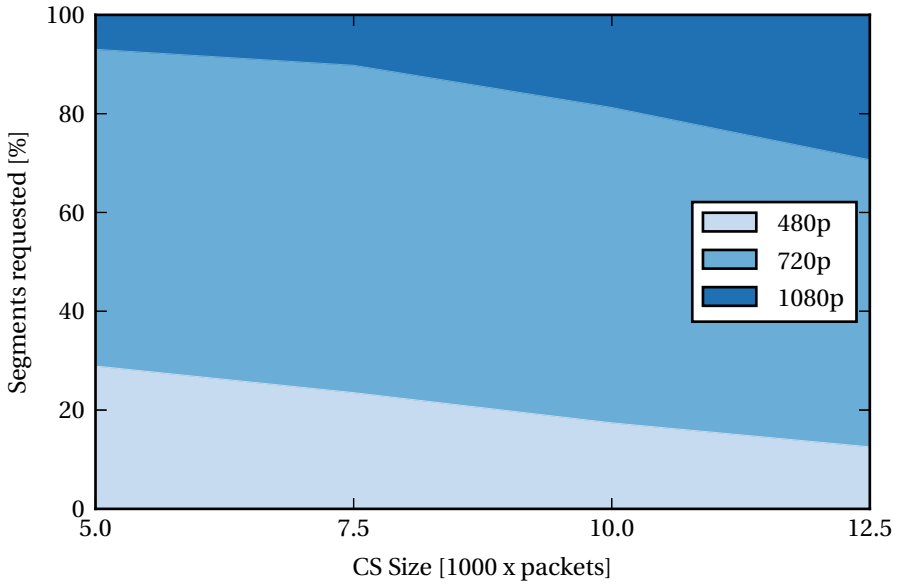

Figure 5.9: Percentage of video segments delivered in each of the available representations, with the LCE+LRU caching policy.

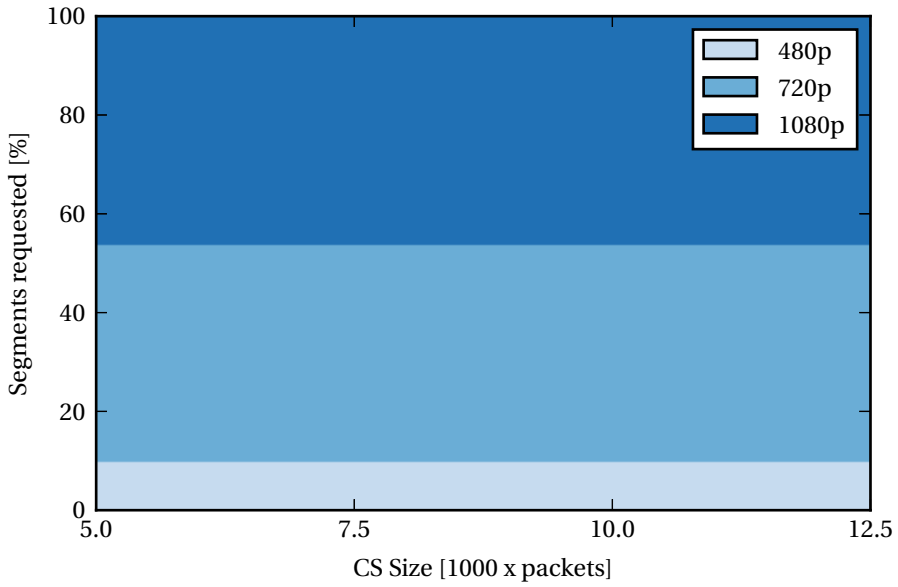

Figure 5.10: Percentage of video segments delivered in each of the available representations, with the LCE+NoLimit caching policy.

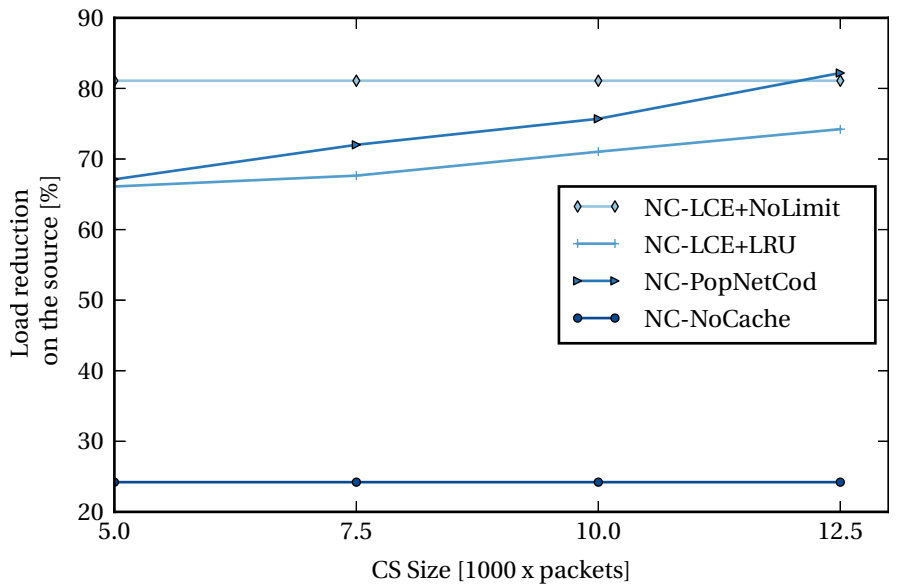

Figure 5.11: Load reduction in the source, measured as the percentage of Data packets delivered to the clients not sent by the source.

total number of Data packets received by all the clients. In Fig. 5.11 we can see that by using the PopNetCod caching policy, the source load is reduced by up to 10% more than by using the LCE+LRU caching policy.

5.6 Conclusion

In this Chapter we have presented PopNetCod, a popularity-based caching policy for data intensive applications communicating over network coding enabled NDN. PopNetCod is a distributed caching policy, where each router aims at increasing its local cache-hit rate, by measuring the popularity of each content object and using it to determine the number of Data packets for each content object that it caches in its content store. PopNetCod takes the cache placement decisions when Interests arrive at the routers, which naturally enables edge caching. We have implemented the proposed caching policy by modifying the codebase of the NetCodNDN architecture to enable the use of caching policies, in particular the PopNetCod caching policy. The evaluation of the PopNetCod caching policy is performed in a Netflix-like video streaming scenario. The results show that, in comparison with a caching policy that uses the Leave Copy Everywhere (LCE) placement policy and the Least Recently Used (LRU) eviction policy, PopNetCod achieves a higher cache-hit rate, closer to the results obtained in Chapter 4 with a content store with unlimited capacity. The increased cache-hit rate reduces the number of Interests that the source should satisfy, and also increases the goodput seen by the clients. Thus, our caching policy presents benefits for the content providers, by reducing the load that its servers receive and hence its operative costs, and for the end-users, which are able to watch higher quality videos.

6

Conclusions

The fast and huge increase of Internet traffic due to data intensive applications motivates the development of new communication methods that can deal with the growing volume of data traffic. To this aim, Named Data Networking (NDN) has been proposed as a future Internet architecture that enables ubiquitous in-network caching and naturally supports multi-path data delivery. This thesis proposed to integrate network coding into the NDN architecture, to improve the performance of data intensive applications in networks with limited bandwidth, network bottlenecks and packet losses. Overall, network coding brings significant benefits to data intensive applications over NDN. All the nodes in the content retrieval process, *i.e.*, the clients, the intermediate routers and the sources, could benefit from the use of network coding. The clients are able to retrieve content faster, the intermediate routers are able to serve more content from the edge of the network, reducing the load in the core routers, and the sources receive less requests, which reduces its processing and bandwidth needs. In the next Sections we first describe the main contributions of this thesis. Then, we propose future directions for network coding enabled NDN architectures.

6.1 Main Contributions

In order to study and quantify the benefits that network coding brings to data intensive applications in NDN, an architecture that integrates network coding into NDN was needed. Due to the lack of well defined and data intensive oriented network coding enabled NDN architectures in the literature, the first contribution of this thesis is NetCodNDN, an architecture that integrates network coding into NDN, presented in Chapter 3. In comparison to previous works proposing to enable network coding in NDN, NetCodNDN permits Interest aggregation and Interest pipelining, which reduce the data retrieval times in data intensive applications. The evaluation showed that the NetCodNDN architecture leads to significant improvements in terms of content retrieval delay, compared to the original NDN. The performance gains were verified for content retrieval in various network scenarios.

Having an architecture efficiently integrates network coding into NDN for data intensive applications, the next contribution of this thesis related to analyzing the benefits that network coding brings to data intensive applications in NDN. Since video streaming is the most popular data intensive application, this thesis used dynamic adaptive video streaming as a data intensive application to study. Thus, Chapter 4 proposed a Dynamic Adaptive Streaming over NetCodNDN (DAS-NetCodNDN) architecture. In comparison to previous works proposing dynamic adaptive streaming over NDN, DAS-NetCodNDN exploits the benefits that network coding brings to video streaming applications, in multi-source and multi-client scenarios. The experimental evaluation has been performed in a Netflix-like scenario. The results showed that the use of network coding permits to exploit more efficiently the available network resources, which leads to reduced data traffic load on the video sources, increased cache-hit rate at the in-network caches and faster adaptation of the requested video quality by the clients.

Having experimentally demonstrated and quantified the benefits that network coding brings to data intensive applications over NDN, the final contribution of this thesis was to limit the capacity of the content stores (*i.e.*, caches) available at the intermediate routers. When the content store capacity is limited, routers need a caching policy that decides which Data packets are cached and which ones are evicted when the content store capacity is reached. To this aim, Chapter 5 proposed PopNetCod, a popularity-based caching policy for data intensive applications communicating over NetCodNDN. PopNetCod is a distributed caching policy, where each router aims at increasing its

local cache-hit rate, by measuring the popularity of each content object and using it to determine the number of Data packets for each content object that it caches in its content store. The design of the PopNetCod caching policy favors edge caching, which has been demonstrated to be beneficial. The evaluation of the PopNetCod caching policy has been performed in a Netflix-like video streaming scenario, similar to the one presented in Chapter 4. The results showed that, in comparison with a caching policy that uses the Leave Copy Everywhere (LCE) placement policy and the Least Recently Used (LRU) eviction policy, PopNetCod achieves a higher cache-hit rate at the intermediate routers. The increased cache-hit rate reduces the number of Interests that the source should satisfy, and also increases the goodput seen by the clients. Thus, our caching policy presents benefits for the content providers, by reducing the load that its servers receive and hence its operative costs, and for the end-users, which are able to watch higher quality videos.

Additionally, this thesis also provided a complete codebase of a network coding enabled NDN architecture. The provided codebase uses open-source libraries and follows the NDN project codebase guidelines, to simplify its re-usability and expansion. In particular, we used the NDN Forwarding Daemon (NFD) codebase [61] as a base for the NetCodNDN architecture. To enable network coding, we have used Kodo [69], an industry standard C++ library that implements network coding functionality in an optimized way. Moreover, we have used the Adaptive Multimedia Streaming with ndnSIM (AMuSt) codebase [43] and the libdash library [60] for the DAS-NetCodNDN architecture. Currently, libdash is the official reference software of the DASH standard. The provided codebase also permits the development of new caching policies, which can be designed and plugged to the nodes in a simple way.

6.2 Future Directions

While this thesis provided an architecture that enabled network coding on NDN and demonstrated its benefits in different scenarios, many opportunities for extending the scope of this thesis remain. This Section presents some of these future research directions.

In the NetCodNDN architecture, intermediate routers aggregate Interests based on the premise that each Interest will bring back an innovative Data packet, which is not always true. Non-innovative Data packets may be received, and also Interests or Data

packets may be lost. Thus, it would be interesting to study new approaches in which the routers can measure the packet loss rate and the non-innovative Data packet rate, among other metrics, and take them into consideration when performing Interest aggregation, to reduce the decoding delay.

Moreover, when a NetCodNDN router receives a Data packet that is non-innovative, it does not forward it further downstream. The router does this to avoid wasting network resources in the transmission of Data packets that have a high probability of being non-innovative for the clients. However, Interest forwarding strategies could use the non-innovative rate of the different faces to adjust where to forward Interests. A possible way to share this "non-innovativeness" information, without the need to send the complete Data packets, is to use negative acknowledgments (NACK), which have been defined in newer versions of the NDN architecture [4]. NACKs are used by routers to signal downstream nodes that they were unable to satisfy a particular Interest. NACKs carry the Interest that could not be satisfied and a reason why the router was unable to satisfy the Interest. In NetCodNDN, a new reason "non-innovative Data packet received" could be defined for the NACK. Interest forwarding strategies could use NACKs with this reason to reduce the number of Interests they forward to faces over which a high number of NACKs with the reason "non-innovative Data packet received" have been received. Furthermore, the non-innovative rate of a face could also be used to improve the Interest aggregation procedure in NetCodNDN.

The NetCodNDN architecture considers that generations [20] are defined by the source and that they do not vary after being defined. As a consequence, the Data packets that belong to a generation can be cached in the routers and they can be easily re-used to serve future Interests for the same generation. However, the use of generations fixes the delay seen by the client to decode the original content object, as it needs to collect a number of network coded Data packets that is at least similar to the generation size. This is not ideal since the network conditions may vary, making the delay to decode the original content object too large for the application. An alternative to the use of generations is to use sliding-window network coding schemes [40], in which the number of Data packets that could be used to generate a network coded Data packet (*i.e.*, the code block size) is decided dynamically according to the network conditions measured by the sources and/or the clients. However, considering the in-network caching capabilities of NetCodNDN, it is not trivial to re-use a cached sliding-window network coded Data packet, since the Data packets used to generate it may be different to the ones that a later client will request, creating a situation in

which cached network coded Data packets may be useless for subsequent clients.

Security in NetCodNDN is another aspect that needs to improved. In particular, content *confidentiality*, *i.e.*, only the nodes with the right to view a content object are able to decode it, and content *authenticity*, *i.e.*, the retrieved content object is exactly as the original trusted source provided it.

Traditionally, content confidentiality in NDN is provided by end-to-end encryption. The sources encrypt the content objects or even the names or name prefixes of the content objects, and then, the clients use the decryption keys provided by the sources to get the original content object. The use of network coding provides inherent content confidentiality properties, since an external eavesdropper that gathers a few Data packets will not be able to see the original content unless it receives enough Data packets to decode it. An approach that takes profit of these network coding properties to reduce the computational complexity of end-to-end encryption is Secure Practical Network Coding (SPOC) [96]. In this approach, a subset of the encoding coefficients of each Data packet, called the *locked coefficients*, are encrypted with keys that are available at the source and at the client, but not at the routers. The routers can still perform network coding operations in another subset of the encoding coefficients of each Data packets, called the *unlocked coefficients*. The unlocked coefficients do not provide any information for effectively decoding the Data packets without access to a decrypted version of the locked coefficients. An adaptation of SPOC could be integrated into NetCodNDN to enable efficient content confidentiality.

In NDN, content authenticity is provided by signing the content object and its name. The sources sign the Data packets that they store, and the clients and routers can use the generated signature to verify the authenticity of the Data packets. However, in NetCodNDN the intermediate routers are able to generate new network coded Data packets based on the Data packets they receive. This means that the signature created by the source is invalidated and not able to be used for authenticity verification. Gkantsidis *et al.* [32] propose an approach for content signature in network coding enabled P2P networks. This approach proposes to sign the network coded Data packets with homomorphic hash functions, which have the property that the hash value of a linear combination of some input Data packets can be constructed efficiently by a combination of the hashes of the input Data packets. The verification of signatures created with homomorphic hash functions has high computational complexity, which would create high delays if each network node decides to verify all

the Data packets they receive. For that reason, Gkantsidis *et al.* [32] also propose a cooperative approach to reduce the overall hash verification complexity, by allowing the routers to verify a subset of the Data packets they receive, and informing their neighbors of any malicious Data packet detected. Understanding the best way to integrate this approach into the NetCodNDN architecture to enable network coded content authenticity is left as future work.

It is worth noting that the aforementioned security issues of the NetCodNDN architecture are only affect network coded content retrieval. The NetCodNDN architecture also permits content retrieval using the original NDN procedures, in which case, the security measures of the NDN architecture are used.

In this thesis, we considered that the clients' position is fixed, *i.e.*, clients are connected to the network through the same edge routers during the content retrieval process. To fully support content retrieval in mobile scenarios (*e.g.*, VANETs, mobile phones, *etc.*), the NetCodNDN architecture should be enhanced. In particular, in the current NetCodNDN architecture the routers count the number of Data packets that have been sent over a particular face, which in the case of edge routers, counts the number of Data packets sent to a particular client. This information is used to avoid sending the same Data packets twice, and to enable Interest pipelining. However, in a mobile scenario, this information is invalid after a client moves to a new location. As a consequence, the rate of duplicate Data packets received by the client could increase, delaying the content retrieval process. One approach that could be used to alleviate this issue is to enable clients to share information about the Data packets that they have previously received with any new edge router, after moving to a new location. With this information, the edge router could compute the number of Data packets from its content store that are duplicated to the client, and initialize the counters appropriately. However, this solution requires sharing information at the beginning of every new connection, which increases the overhead of the architecture. New solutions that improve the performance of the NetCodNDN architecture in mobile scenarios should be studied.

Finally, the DAS-NetCodNDN architecture proposed in this thesis for adaptive video streaming over NetCodNDN considers a Video-on-Demand (VoD) scenario, in which clients request the video segments at different times. An interesting expansion to the DAS-NetCodNDN architecture would be to consider the use of live video streaming scenarios, where the Data packets are valid for a short period of time, and where

Interest aggregation plays a more important role than caching. HTTP Live Streaming (HLS) [66] is a well stablished approach that resemble the Dynamic Adaptive Streaming over HTTP (DASH) [37] approach in which DAS-NetCodNDN is based on. Thus, it would be interesting to extend DAS-NetCodNDN to allow HLS, and understand the benefits that network coding brings to live video streaming over NDN.

Bibliography

[1] 3GPP, "3GPP TS 26.346 V7.1.0, Technical Specification Group Services and System Aspects; Multimedia Broadcast/Multicast Service; Protocols and Codecs," Jun. 2005.

[2] A. Aaron, Z. Li, M. Manohara, J. D. Cock, and D. Ronca, "The Netflix tech blog: Per-title encode optimization," http://techblog.netflix.com/2015/12/per-title-encode-optimization.html, Dec. 2015.

[3] N. Abani, G. Farhadi, A. Ito, and M. Gerla, "Popularity-based partial caching for information centric networks," in *Proc. Med-Hoc-Net'16*, Jun. 2016, pp. 1–8.

[4] A. Afanasyev *et al.*, "NFD developer's guide," Named Data Networking project, Technical Report NDN-0021 Revision 7, Oct. 2016.

[5] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, Jul. 2012.

[6] B. Ahlgren, M. D'Ambrosio, M. Marchisio, I. Marsh, C. Dannewitz, B. Ohlman, K. Pentikousis, O. Strandberg, R. Rembarz, and V. Vercellone, "Design considerations for a network of information," in *Proc. of ACM CoNEXT'08*, 2008, pp. 1–6.

[7] R. Ahlswede, N. Cai, S.-Y. Li, and R. Yeung, "Network information flow," *IEEE Trans. Information Theory*, vol. 46, no. 4, pp. 1204–1216, Jul. 2000.

[8] C. Anastasiades, N. Thomos, A. Striffeler, and T. Braun, "RC-NDN: Raptor codes enabled named data networking," in *Proc. of IEEE ICC'15*, Jun. 2015, pp. 3026–3032.

[9] R. Bassoli, H. Marques, J. Rodriguez, K. W. Shum, and R. Tafazolli, "Network coding theory: A survey," *IEEE Communications Surveys Tutorials*, vol. 15, no. 4, pp. 1950–1978, Apr. 2013.

[10] C. Bernardini, T. Silverston, and O. Festor, "MPC: Popularity-based caching strategy for content centric networks," in *Proc. of IEEE ICC'13*, Jun. 2013, pp. 3619–3623.

[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. of the ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[12] T. Böttger, F. Cuadrado, G. Tyson, I. Castro, and S. Uhlig, "Open connect everywhere: a glimpse at the internet ecosystem through the lens of the netflix cdn," *arXiv preprint arXiv:1606.05519*, Jun. 2016.

[13] E. Bourtsoulatze, N. Thomos, and P. Frossard, "Distributed rate allocation in inter-session network coding," *IEEE Trans. Multimedia*, vol. 16, no. 6, pp. 1752–1765, Oct. 2014.

[14] E. Bourtsoulatze, N. Thomos, J. Saltarin, and T. Braun, "Content-aware delivery of scalable video in network coding enabled named data networks," *IEEE Trans. on Multimedia*, vol. PP, no. 99, Aug. 2017.

[15] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege, "A digital fountain approach to reliable distribution of bulk data," *SIGCOMM Comput. Commun. Rev.*, vol. 28, pp. 56–67, Oct. 1998.

[16] "CCNDC(1) Manual Page, Project CCNx ®, version 0.8.2," https://github.com/ProjectCCNx/ccnx/blob/master/doc/manpages/ccndc.1.txt.

[17] "Project CCNx ®, version 0.8.2," http://www.ccnx.org/releases/ccnx-0.8.2/doc/.

[18] V. Cerf and R. Kahn, "A protocol for packet network intercommunication," *IEEE Trans. on Communications*, vol. 22, no. 5, pp. 637–648, May 1974.

[19] K. Cho, M. Lee, K. Park, T. T. Kwon, Y. Choi, and S. Pack, "WAVE: Popularity-based and collaborative in-network caching for content-oriented networks," in *Proc. of IEEE INFOCOM'12 Workshops*, Mar. 2012, pp. 316–321.

[20] P. Chou and Y. Wu, "Network coding for the Internet and wireless networks," *IEEE Signal Processing Magazine*, vol. 24, no. 5, pp. 77–85, Sep. 2007.

[21] "Cisco Visual Networking Index: Forecast and Methodology, 2016-2021," White Paper, Cisco Systems Inc., Jun. 2016.

[22] N. Cleju, N. Thomos, and P. Frossard, "Selection of network coding nodes for minimal playback delay in streaming overlays," *IEEE Trans. on Multimedia*, vol. 13, no. 5, pp. 1103–1115, Oct. 2011.

[23] A. Dabirmoghaddam, M. Dehghan, and J. J. Garcia-Luna-Aceves, "Characterizing interest aggregation in content-centric networks," in *Proc. of IFIP Networking'16*, May 2016, pp. 449–457.

[24] A. Dabirmoghaddam, M. Mirzazad-Barijough, and J. J. Garcia-Luna-Aceves, "Understanding Optimal Caching and Opportunistic Caching at "the Edge" of Information-Centric Networks," in *Proc. of ACM ICN'14*, 2014, pp. 47–56.

[25] A. Detti, M. Pomposini, N. Blefari-Melazzi, S. Salsano, and A. Bragagnini, "Offloading cellular networks with information-centric networking: the case of video streaming," in *Proc. of IEEE WoWMoM'12*, San Francisco, CA, USA, Jun. 2012.

[26] S. K. Fayazbakhsh, Y. Lin, A. Tootoonchian, A. Ghodsi, T. Koponen, B. Maggs, K. Ng, V. Sekar, and S. Shenker, "Less pain, most of the gain: incrementally deployable ICN," in *Proc. of ACM SIGCOMM'13*, 2013, pp. 147–158.

[27] R. Fielding and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content," RFC 7231, IETF, Jun. 2014. [Online]. Available: https://tools.ietf.org/rfc/rfc7231.txt

[28] A. Ford *et al.*, "TCP extensions for multipath operation with multiple addresses," RFC 6824, IETF, Jan. 2013. [Online]. Available: http://www.rfc-editor.org/rfc/rfc6824.txt

[29] N. Fotiou, D. Trossen, and G. C. Polyzos, "Illustrating a publish-subscribe internet architecture," *Telecommunication Systems*, vol. 51, no. 4, pp. 233–245, Dec 2012.

[30] C. Fragouli, E. Soljanin *et al.*, "Network coding applications," *Foundations and Trends® in Networking*, vol. 2, no. 2, pp. 135–269, 2008.

[31] R. Gallager, "Low-density parity-check codes," *IRE Trans. on Information Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.

[32] C. Gkantsidis and P. R. Rodriguez, "Cooperative security for network coding file distribution," in *Proc. of IEEE INFOCOM'06*, Apr. 2006.

[33] A. Gomes and T. Braun, "Load balancing in LTE mobile networks with Information-Centric Networking," in *Proc. of ICC'15 Workshops*, Jun. 2015, pp. 1845–1851.

[34] T. Ho, R. Koetter, M. Medard, D. R. Karger, and M. Effros, "The benefits of coding over routing in a randomized setting," in *Proc. of IEEE ISIT'03*, Jun. 2003, pp. 442–.

[35] T. Ho, M. Medard, J. Shi, M. Effros, and D. R. Karger, "On randomized network coding," in *Proc. of Allerton'03*, Oct. 2003.

[36] ICNRG, "Design Choices and Differences for NDN and CCNx 1.0 Implementations of Information-Centric Networking," draft-icnrg-harmonization-00, ICNRG, Jul. 2017. [Online]. Available: https://icnrg.github.io/draft-icnrg-harmonization/draft-icnrg-harmonization-00.html

[37] ISO/IEC JTC 1/SC 29, *Information technology – Dynamic adaptive streaming over HTTP (DASH) – Part 1: Media presentation description and segment formats*, ISO/IEC 23 009-1:2014, May 2014.

[38] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proc. of ACM CoNEXT'09*, Rome, Italy, Dec. 2009, pp. 1–12.

[39] S. Jaggi, P. Sanders, P. Chou, M. Effros, S. Egner, K. Jain, and L. Tolhuizen, "Polynomial Time Algorithms for Multicast Network Code Construction," *IEEE Trans. on Information Theory*, vol. 51, no. 6, pp. 1973 – 1982, Jun. 2005.

[40] M. Karzand and D. J. Leith, "Low delay random linear coding over a stream," in *Proc. of Allerton'14*, Sep. 2014, pp. 521–528.

[41] R. Koetter and M. Médard, "An algebraic approach to network coding," *IEEE/ACM Trans. Networking*, vol. 11, no. 5, pp. 782–795, Oct. 2003.

[42] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. of SIGCOMM'07*, 2007, pp. 181–192.

[43] C. Kreuzberger, D. Posch, and H. Hellwagner, "AMuSt Framework - Adaptive Multimedia Streaming Simulation Framework for ns-3 and ndnSIM," https://github.com/ChristianKreuzberger/amust-simulator, 2016.

[44] N. Laoutaris, H. Che, and I. Stavrakakis, "The lcd interconnection of lru caches and its analysis," *Performance Evaluation*, vol. 63, no. 7, pp. 609 – 634, 2006.

[45] S. Lederer, C. Mueller, B. Rainer, C. Timmerer, and H. Hellwagner, "Adaptive streaming over content centric networks in mobile networks using multiple links," in *Proc. of IEEE ICC'13 Workshops*, Jun. 2013, pp. 677–681.

[46] ——, "An experimental analysis of dynamic adaptive streaming over HTTP in content centric networks," in *Proc. of IEEE ICME'13*, San Jose, CA, USA, Jul. 2013, pp. 1–6.

[47] S. Li, J. Xu, M. van der Schaar, and W. Li, "Popularity-driven content caching," in *Proc. of IEEE INFOCOM'16*, Apr. 2016, pp. 1–9.

[48] Z. Li, H. Xu, and B. Li, "Network coding in bi-directed and peer-to-peer networks," in *Next-Generation Internet: Architectures and Protocols*, ch. 17.

[49] J. Llorca, A. Tulino, K. Guan, and D. Kilper, "Network-coded caching-aided multicast for efficient content delivery," in *Proc. of IEEE ICC'13*, Budapest, Hungary, Jun. 2013, pp. 3557–3562.

[50] M. Luby, "Lt codes," in *Proc. of IEEE FOCS'02*, 2002.

[51] M. Luby, A. Shokrollahi, M. Watson, T. Stockhammer, and L. Minder, "RaptorQ Forward Error Correction Scheme for Object Delivery," RFC 6330, IETF, Aug. 2011. [Online]. Available: https://tools.ietf.org/rfc/rfc6330.txt

[52] M. Luby, "Tornado codes: Practical erasure codes based on random irregular graphs," in *Proc. of RANDOM'98*, 1998, pp. 171–171.

[53] D. E. Lucani, M. V. Pedersen, J. Heide, and F. H. P. Fitzek, "Fulcrum Network Codes: A Code for Fluid Allocation of Complexity," *arXiv preprint arXiv:1404.6620*, Nov. 2015.

[54] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the Evolution of ndnSIM: an Open-Source Simulator for NDN Experimentation," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, Jul. 2017.

[55] K. Matsuzono, H. Asaeda, and T. Turletti, "Low latency low loss streaming using in-network coding and caching," in *Proc. of IEEE INFOCOM'17*, Atlanta, USA, May 2017.

[56] M. Mitzenmacher, "Digital fountains: a survey and look forward," in *Proc. of IEEE ITW'04*, 2004, pp. 271 – 276.

[57] P. Mockapetris, "Domain names - concepts and facilities," RFC 1034, IETF, Nov. 1987. [Online]. Available: https://tools.ietf.org/rfc/rfc1034.txt

[58] ——, "Domain names - implementation and specification," RFC 1035, IETF, Nov. 1987. [Online]. Available: https://tools.ietf.org/rfc/rfc1035.txt

[59] M.-J. Montpetit, C. Westphal, and D. Trossen, "Network coding meets information-centric networking: an architectural case for information dispersion through native network coding," in *Proc. of ACM NoM'12 Workshop*, Hilton Head, South Carolina, USA, Jun. 2012, pp. 31–36.

[60] C. Mueller, S. Lederer, J. Poecher, and C. Timmerer, "Demo paper: libdash - an open source software library for the MPEG-DASH standard," in *Proc. of IEEE ICME'13 Workshops*, San Jose, CA, USA, Jul. 2013.

[61] Named Data Networking (NDN) Project, "Named Data Networking Forwarding Daemon," https://github.com/named-data/NFD.

[62] "The Netflix ISP Speed Index," Netflix Inc., Dec. 2016. [Online]. Available: https://ispspeedindex.netflix.com/

[63] "The network simulator - ns3," http://www.nsnam.org/.

[64] "Direct Code Execution (DCE)," https://www.nsnam.org/overview/projects/direct-code-execution/.

[65] J. Ozer, "What is AV1?" http://www.streamingmedia.com/Articles/Editorial/What-Is-.../What-is-AV1-111497.aspx.

[66] R. Pantos and W. May, "HTTP Live Streaming," draft-pantos-http-live-streaming-23, IETF, May 2017. [Online]. Available: https://tools.ietf.org/html/draft-pantos-http-live-streaming-23

[67] G. Parisis, V. Sourlas, K. V. Katsaros, W. K. Chai, and G. Pavlou, "Enhancing multi-source content delivery in content-centric networks with fountain coding," in *Proc. of CCDWN'16 Workshop*, 2016, pp. 1–7.

[68] J. Pearl, "Reverend bayes on inference engines: A distributed hierarchical approach," in *Proc. of AAAI'82*, 1982, pp. 133–136.

[69] M. Pedersen, J. Heide, and F. H. P. Fitzek, "Kodo: an open and research oriented network coding library," in *Proc. of IFIP Networking'11*, Valencia, Spain, May 2011, pp. 145–152.

[70] M. Pedersen, J. Heide, P. Vingelmann, and F. Fitzek, "Network coding over the $2^{32} - 5$ prime field," in *Proc. of IEEE ICC'13*, Budapest, Hungary, Jun. 2013.

[71] K. Pentikousis *et al.*, "Information-Centric Networking: Baseline Scenarios," RFC 7476, Mar. 2015. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7476.txt

[72] "PlanetLab," https://www.planet-lab.org/.

[73] D. Posch, C. Kreuzberger, B. Rainer, and H. Hellwagner, "Using in-network adaptation to tackle inefficiencies caused by dash in information-centric networks," in *Proc. of VideoNext '14 Workshop*, 2014, pp. 25–30.

[74] I. Psaras, W. K. Chai, and G. Pavlou, "In-network cache management and resource allocation for information-centric networks," *IEEE Trans. on Parallel and Distributed Systems*, vol. 25, no. 11, pp. 2920–2931, Nov. 2014.

[75] A. Ramakrishnan, C. Westphal, and J. Saltarin, "Adaptive video streaming over ccn with network coding for seamless mobility," in *Proc. of IEEE ISM'16*, Dec. 2016.

[76] V. Roca, Z. Khallouf, and J. Laboure, "Design and Evaluation of a Low Density Generator Matrix (LDGM) Large Block FEC Codec," in *Proc. of COST 264 NGC'03*, Sep. 2003, pp. 193–204.

[77] G. Rossini and D. Rossi, "Evaluating CCN multi-path interest forwarding strategies," *Computer Communications*, vol. 36, no. 7, pp. 771 – 778, Apr. 2013.

[78] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "NetCodCCN: a network coding approach for content-centric networks," in *Proc. of IEEE INFOCOM'16*, Apr. 2016.

[79] J. Saltarin, T. Braun, E. Bourtsoulatze, and N. Thomos, "Popnetcod: A popularity-based caching policy for network coding enabled named data networks," in *under submission*, Jul. 2017.

[80] J. Saltarin, E. Bourtsoulatze, N. Thomos, and T. Braun, "Adaptive video streaming with network coding enabled named data networking," *IEEE Trans. on Multimedia*, vol. 19, no. 10, Aug. 2017.

[81] K. M. Schneider and U. R. Krieger, "Beyond network selection: exploiting access network heterogeneity with named data networking," in *Proc. of ACM ICN'15*, San Francisco, USA, Sep. 2015, pp. 137–146.

[82] A. M. Sheikh, A. Fiandrotti, and E. Magli, "Distributed scheduling for low-delay and loss-resilient media streaming with network coding," *IEEE Trans. Multimedia*, vol. 16, no. 8, pp. 2294–2306, Dec. 2014.

[83] A. Shokrollahi, "Raptor codes," *IEEE Trans. on Information Theory*, vol. 52, no. 6, pp. 2551 –2567, Jun. 2006.

[84] M. Sipos, J. Heide, D. Lucani, M. Pedersen, F. Fitzek, and H. Charaf, "Adaptive network coded clouds: High speed downloads and cost-effective version control," *IEEE Trans. on Cloud Computing*, vol. PP, no. 99, 2017.

[85] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, Apr. 2011.

[86] G. J. Sullivan, J. R. Ohm, W. J. Han, and T. Wiegand, "Overview of the high efficiency video coding (hevc) standard," *IEEE Trans. on Circuits and Systems for Video Technology*, vol. 22, no. 12, pp. 1649–1668, Dec. 2012.

[87] Y. Sun, S. K. Fayaz, Y. Guo, V. Sekar, Y. Jin, M. A. Kaafar, and S. Uhlig, "Trace-driven analysis of ICN caching algorithms on video-on-demand workloads," in *Proc. of ACM CoNEXT '14*, 2014, pp. 363–376.

[88] J. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: theory and implementation," *Proc. of the IEEE*, vol. 99, no. 3, pp. 490–512, Mar. 2011.

[89] N. Thomos and P. Frossard, "Toward one Symbol Network Coding Vectors," *IEEE Communications letters*, vol. 16, no. 11, pp. 1860–1863, Nov. 2012.

[90] N. Thomos, E. Kurdoglu, P. Frossard, and M. van der Schaar, "Adaptive prioritized random linear coding and scheduling for layered data delivery from multiple servers," *IEEE Trans. Multimedia*, vol. 17, no. 6, pp. 893–906, Jun. 2015.

[91] N. Thomos and P. Frossard, "Media coding for streaming in networks with source and path diversity," in *Intelligent Multimedia Transmission: Techniques and Applications*. Springer-Verlag, 2009.

[92] M. Thomson, E. Damaggio, and B. Raymor, "Generic Event Delivery Using HTTP Push," RFC 8030, IETF, Dec. 2016. [Online]. Available: https://tools.ietf.org/rfc/rfc8030.txt

[93] C. Timmerer, M. Maiero, and B. Rainer, "Which adaptation logic? An objective and subjective performance evaluation of http-based adaptive media streaming systems," *arXiv preprint arXiv:1606.00341*, Jun. 2016.

[94] D. Trossen and G. Parisis, "Designing and realizing an information-centric internet," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 60–67, Jul. 2012.

[95] C. Tsilopoulos, G. Xylomenos, and G. C. Polyzos, "Are information-centric networks video-ready?" in *Proc. of IEEE PV'13 Workshop*, Dec. 2013, pp. 1–8.

[96] J. P. Vilela, L. Lima, and J. Barros, "Lightweight security for network coding," in *Proc. of IEEE ICC'08*, May 2008, pp. 1750–1754.

[97] J. Wang, J. Ren, K. Lu, J. Wang, S. Liu, and C. Westphal, "An Optimal Cache Management Framework for Information-Centric Networks with Network Coding," in *Proc. of IFIP NETWORKING'14*, Trondheim, Norway, Jun. 2014.

[98] X. Wang, J. Ren, T. Tong, R. Dai, S. Xu, and S. Wang, "Towards efficient and lightweight collaborative in-network caching for content centric networks," in *Proc. of IEEE GLOBECOM'16*, Dec. 2016, pp. 1–7.

[99] C. Westphal *et al.*, "Adaptive video streaming over information-centric networking (ICN)," RFC 7933, IRTF, Aug. 2016. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7933.txt

[100] Q. Wu, Z. Li, and G. Xie, "CodingCache: multipath-aware CCN cache with network coding," in *Proc. of ACM ICN'2013 Workshop*, Hong Kong, China, Aug. 2013, pp. 41–42.

[101] Y. Wu, P. Chou, and K. Jain, "A comparison of network coding and tree packing," in *Proc. of IEEE ISIT'04*, Chicago, IL, USA, Jun. 2004, p. 145.

[102] G. Xylomenos, C. N. Ververidis, V. A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. V. Katsaros, and G. C. Polyzos, "A survey of information-centric networking research," *IEEE Communications Surveys Tutorials*, vol. 16, no. 2, pp. 1024–1049, Feb. 2014.

[103] E. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu, and D. Leong, "Vip: A framework for joint dynamic forwarding and caching in named data networks," in *Proc. of ACM ICN'14*, 2014, pp. 117–126.

[104] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," *SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[105] L. Zhang *et al.*, "Named Data Networking (NDN) Project," Named Data Networking project, Technical Report NDN-0001 Revision 7, Oct. 2010.

[106] M. Zhang, H. Li, F. Chen, H. Hou, H. An, W. Wang, and J. Huang, "A general co/decoder of network coding in hdl," in *Proc. of NetCod'11*, Beijing, China, Jul. 2011.

Declaration of consent

on the basis of Article 28 para. 2 of the RSL05 phil.-nat.

Name/First Name: Saltarin De Arco, Jonnahtan Eduardo

Matriculation Number: 13-128-970

Study program: Computer Science

Bachelor ☐ Master ☐ Dissertation ☑

Title of the thesis: Network Coding Enabled Named Data Networking Architectures

Supervisor: Prof. Dr. Torsten Braun

I declare herewith that this thesis is my own work and that I have not used any sources other than those stated. I have indicated the adoption of quotations as well as thoughts taken from other authors as such in the thesis. I am aware that the Senate pursuant to Article 36 para. 1 lit. r of the University Act of 5 September, 1996 is authorised to revoke the title awarded on the basis of this thesis. I allow herewith inspection in this thesis.

Place/Date

Signature

Jonnahtan Saltarin

Curriculum Vitae

Worblentalstrasse 63
CH-3063 Ittigen
+41 78 652 09 69
saltarinj@gmail.com
jonnahtan.me

Education

09.2013 - 10.2017 **PhD, Computer Science**, *University of Bern*, Switzerland.

09.2008 - 04.2011 **MSc, Computer Networks Engineering**, *Polytechnic University of Turin*, Italy.
GPA: 107/110 (5.8/6.0).

09.2003 - 08.2008 **BSc, Electrical Engineering**, *Central University of Venezuela*, Venezuela.
GPA: 16/20 (4.8/6.0), Top 5%.

Professional Experience

09.2013 - 09.2017 **Research Assistant**.
University of Bern, Switzerland – with Prof. Torsten Braun

01.2012 - 09.2013 **Software Engineer & Research Assistant**.
COPElabs, University Lusófona, Portugal – with Profs. Rute Sofia and Paulo Mendes

09.2011 - 01.2012 **Software Engineering Intern**.
Fiat SpA, Italy

01.2011 - 05.2011 **Research Intern**.
EPFL – Swiss Federal Institute of Technology Lausanne, Switzerland – with Prof. Karl Aberer

04.2010 - 12.2010 **Research Intern – MSc Thesis**.
EPFL – Swiss Federal Institute of Technology Lausanne, Switzerland – with Prof. Pascal Frossard

Publications in this Thesis

[1] Jonnahtan Saltarin, Torsten Braun, Eirina Bourtsoulatze, and Nikolaos Thomos. PopNetCod: A Popularity-based Caching Policy for Network Coding enabled Named Data Networking. *Under submission*.

[2] Jonnahtan Saltarin, Eirina Bourtsoulatze, Nikolaos Thomos, and Torsten Braun. Adaptive Video Streaming with Network Coding Enabled Named Data Networking. In *IEEE Transactions on Multimedia (TMM)*, vol. 19, no. 10.

[3] Jonnahtan Saltarin, Eirina Bourtsoulatze, Nikolaos Thomos, and Torsten Braun. NetCodCCN: a Network Coding approach for Content Centric Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, April 2016.